# Multi-Modal Verification of Distributed Systems in Lean

Ilya Sergey

ilyasergey.net


joint work with George Pîrlea, Qiyuan Zhao, Ziyu Mao, Vladimir Gladshtein, and Elad Kinsbruner

# A Framework for Automated and Interactive Verification of Distributed Protocols

# Distributed Protocols

Define how multiple parties (nodes) collaborate with each other to achieve a common goal 🤝

Operating Systems — R. Stockton Gaines Editor

**Time, Clocks, and the Ordering of Events in a Distributed System**

Leslie Lamport
Massachusetts Computer Associates, Inc.

Operating Systems — R. Stockton Gaines Editor

**An Optimal Algorithm for Mutual Exclusion in Computer Networks**

Glenn Ricart
National Institutes of Health

Ashok K. Agrawala
University of Maryland

Minister

Anne, are you prepared to commit to this relationship?

Henry

Anne



*Two-Phase Commit* depiction from Leslie Lamport's video course on TLA+

# Distributed Protocols

Some nodes might *fail* 💥

**Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems**

Brian M. Oki
Barbara H. Liskov

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, MA 02139

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

**There Is More Consensus in Egalitarian Parliaments**

Iulian Moraru, David G. Andersen, Michael Kaminsky
*Carnegie Mellon University and Intel Labs*

**In Search of an Understandable Consensus Algorithm**

Diego Ongaro and John Ousterhout, *Stanford U*

https://www.usenix.org/conference/atc14/technical-sessions/pres

**Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering**

Jialin Li Ellis Michael Naveen Kr. Sharma Adriana Szekeres Dan R. K. Ports
*University of Washington*
{lijl, emichael, naveenks, aaasz, drkp}@cs.washington.edu

# Distributed Protocols

Some nodes might behave *maliciously* 😈

**The Byzantine Generals Problem**

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

**Practical Byzantine Fault Tolerance**

Miguel Castro and Barbara Liskov

**The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus**

DAVID MAZIÈRES, Stellar Development Foundation

**HotStuff: BFT Consensus with Linearity and Responsiveness**

Maofan Yin, Cornell University, VMware Research
Dahlia Malkhi, VMware Research
Michael K. Reiter, UNC-Chapel Hill
Guy Golan Gueta, VMware Research
Ittai Abraham, VMware Research

**All You Need is DAG**

Eleftherios Kokoris-Kogias
IST Austria and Novi Research

Alexander Spiegelman
Novi Research

**Bullshark: DAG BFT Protocols Made Practical**

Alexander Spiegelman
sasha.spiegelman@gmail.com

Neil Giridharan
giridhn@berkeley.edu

**Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback**

Rati Gelashvili, *Novi Research*
Lefteris Kokoris-Kogias, *Novi Research & IST Austria*
Alberto Sonnino, *Novi Research*
Alexander Spiegelman, *Novi Research*
Zhuolun Xiang*, *University of Illinois at Urbana-Champaign*

**A Secure Sharding Protocol For Open Blockchains**

Loi Luu, National University of Singapore, loiluu@comp.nus.edu.sg
Viswesh Narayanan, National University of Singapore, visweshn@comp.nus.edu.sg
Chaodong Zheng, National University of Singapore, chaodong.zheng@comp.nus.edu.sg
Kunal Baweja, National University of Singapore, bawejaku@comp.nus.edu.sg
Seth Gilbert, National University of Singapore, seth.gilbert@comp.nus.edu.sg
Prateek Saxena, National University of Singapore, prateeks@comp.nus.edu.sg

# Verifying Distributed Protocols

Google “Errors found in distributed protocols”
github.com/dranov/protocol-bugs-list

- Safety properties: “Bad thing will not happen.”
- Liveness properties: “Good thing will eventually happen.”

Safety bugs **can** be reliably discovered with conventional black-box **testing**.

# Frameworks for Verifying Distributed Protocols

Proving the Correctness of Disk Paxos in Isabelle/HOL

2005

**IronFleet: Proving Practical Distributed Systems Correct**

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch,
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill

**Ivy: Safety Verification by Interactive Generalization**

Oded Padon
Tel Aviv University, Israel

Kenneth L. McMillan
Microsoft Research, USA

Aurojit Panda
UC Berkeley, USA

**Verdi: A Framework for Implementing and Formally Verifying Distributed Systems**

**Programming and Proving with Distributed Protocols**

Velisarios: Byzantine Fault-Tolerant Protocols Powered by Coq *

**Igloo: Soundly Linking Compositional Refinement and Separation Logic for Distributed System Verification**

CHRISTOPH SPRENGER, TOBIAS KLENZE, MARCO EILERS, FELIX A. WOLF, PETER MÜLLER, MARTIN CLOCHARD, and DAVID BASIN, ETH Zurich, Switzerland

Aneris: A Mechanised Logic for Modular Reasoning about Distributed Systems

lorten Krogh-Jespersen, Amin Timany*, Marit Edna Ohlenbusch,

**LiDO: Linearizable Byzantine Distributed Objects with Refinement-Based Liveness Proofs**

LONGFEI QIU, Yale University, USA
YOONSEUNG KIM, Yale University, USA
JI-YONG SHIN, Northeastern University, USA
JIEUNG KIM, Inha University, South Korea
WOLF HONORÉ*, Yale University, USA
ZHONG SHAO, Yale University, USA

**Compositional Verification of Composite Byzantine Protocols**

Qiyuan Zhao
National University of Singapore
Singapore, Singapore
qiyuanz@comp.nus.edu.sg

George Pîrlea
National University of Singapore
Singapore, Singapore
gpirlea@comp.nus.edu.sg

Karolina Grzeszkiewicz
Yale-NUS College
Singapore, Singapore
karolina.grzeszkiewicz@u.yale-nus.edu.sg

Seth Gilbert
National University of Singapore
Singapore, Singapore
seth.gilbert@comp.nus.edu.sg

Ilya Sergey
National University of Singapore
Singapore, Singapore
ilya@nus.edu.sg

# Frameworks for Verifying Distributed Protocols

Proving the Correctness of Disk Paxos in Isabelle/HOL

2005

**IronFleet: Proving Practical Distributed Systems Correct**

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch,
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill

**Verdi: A Framework for**
**Formally Verifying Dis**

**Programming and Proving wi**

**n by Interactive Generalization**

nneth L. McMillan
crosoft Research, USA

Aurojit Panda
UC Berkeley, USA

**ne Fault-Tolerant Protocols**
**ered by Coq ***


**Igloo: Soundly Linking Composit**
**Separation Logic for Distributed**

CHRISTOPH SPRENGER, TOBIAS KLENZE, M
MÜLLER, MARTIN CLOCHARD, and DAVID BA

**nised Logic for Modular**
**t Distributed Systems**

hin Timany*, Marit Edna Ohlenbusch,

**LiDO: Linearizable Byzantine Dist**
**Refinement-Based Liveness Proofs**

LONGFEI QIU, Yale University, USA
YOONSEUNG KIM, Yale University, USA
JI-YONG SHIN, Northeastern University, USA
JIEUNG KIM, Inha University, South Korea
WOLF HONORÉ*, Yale University, USA
ZHONG SHAO, Yale University, USA

**Composite Byzantine Protocols**

ge Pîrlea
National University of Singapore
Singapore, Singapore
gpirlea@comp.nus.edu.sg

Karolina Grzeszkiewicz
Yale-NUS College
Singapore, Singapore
karolina.grzeszkiewicz@u.yale-nus.edu.sg

National University of Singapore
Singapore, Singapore
qiyuanz@comp.nus.edu.sg

Seth Gilbert
National University of Singapore
Singapore, Singapore
seth.gilbert@comp.nus.edu.sg

Ilya Sergey
National University of Singapore
Singapore, Singapore
ilya@nus.edu.sg

Correctness:
**P**[*no bugs*]

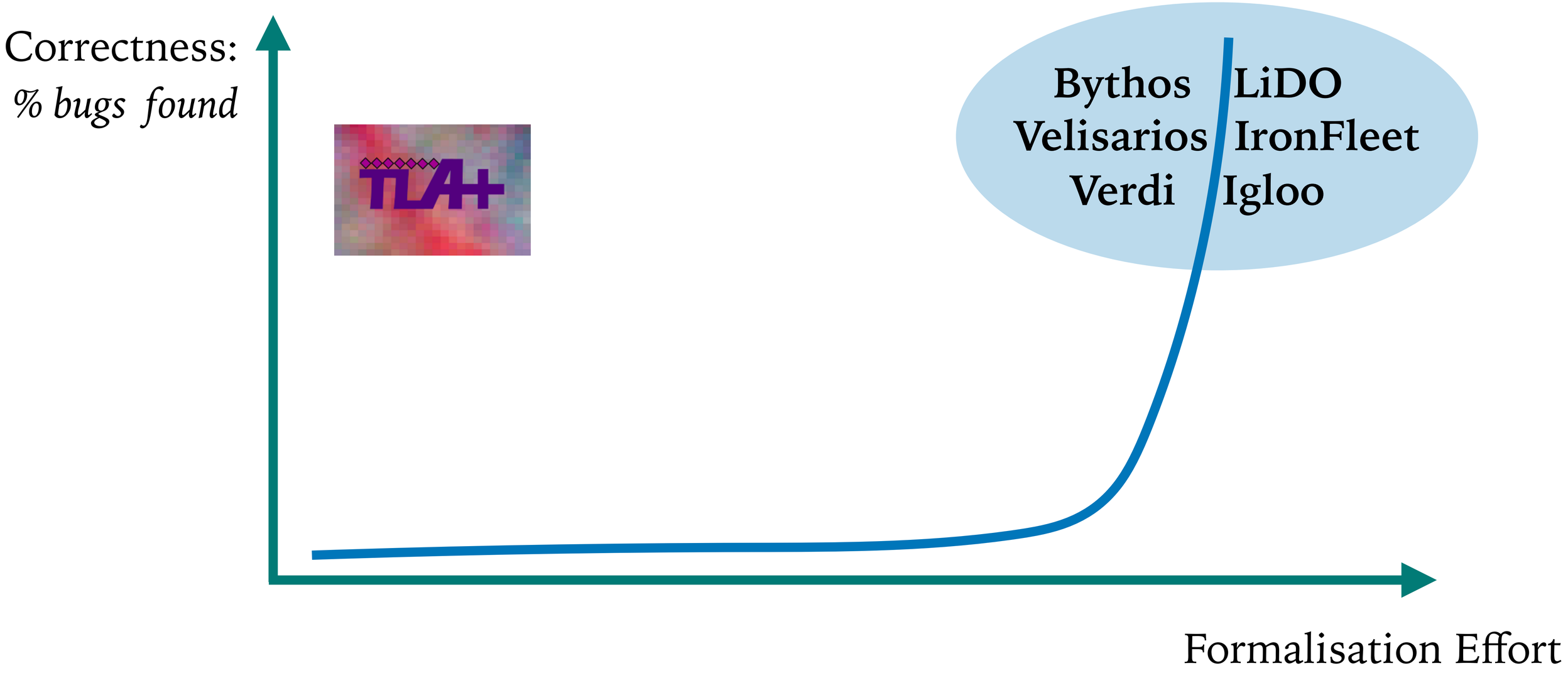

**Bythos** **LiDO**
**Velisarios** **IronFleet**
**Verdi** **Igloo**

Formalisation Effort

Correctness:
*% bugs found*


Bythos LiDO
Velisarios IronFleet
Verdi Igloo

Formalisation Effort


Correctness:
% bugs found
TLA+
Bythos
Velisarios
Verdi
LiDO
IronFleet
Igloo
Formalisation Effort

protocol & properties


✅

or counterexample

Correctness:
*% bugs found*

Bythos LiDo

Formalisation Effort

Correctness:
*% bugs found*

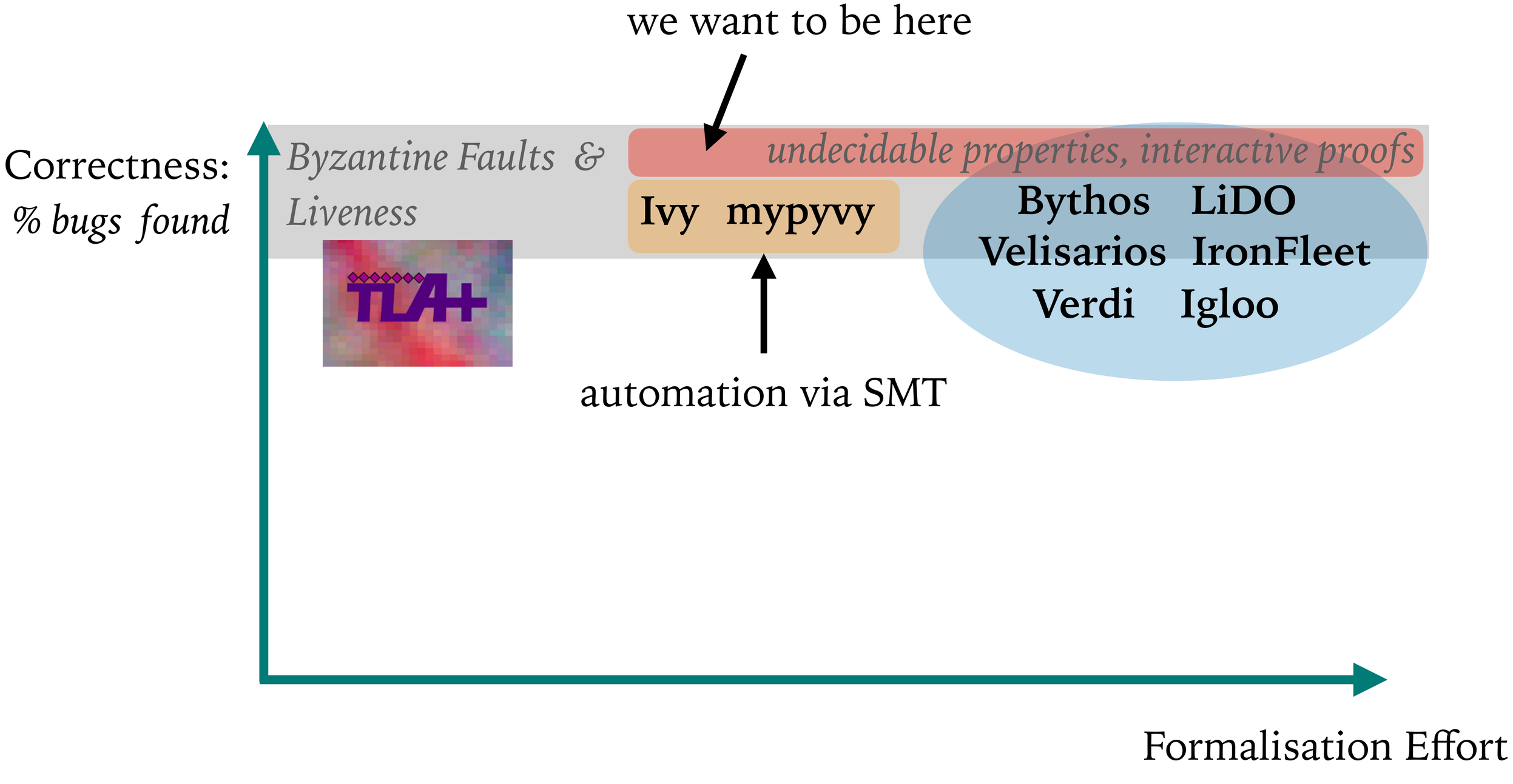
*Byzantine Faults & Liveness*


**Ivy mypyvy**

automation via SMT

**Bythos LiDO**
**Velisarios IronFleet**
**Verdi Igloo**

Formalisation Effort


we want to be here
Correctness:
% bugs found
Byzantine Faults &
Liveness
undecidable properties, interactive proofs
Ivy mypyvy
Bythos LiDO
Velisarios IronFleet
Verdi Igloo
TLA+
automation via SMT
Formalisation Effort

| | Arbitrary Properties | Fast Feedback | Interactive Proofs for Complex (HO) Properties | FOL Proof Automation |
|---|---|---|---|---|
| TLA+ | ✅ | ✅ | 😐 | 😐 |
| Verdi, Grove, Velisarios, Aneris, Bythos, LiDO | ✅ | ❌ | ✅ | ❌ |
| IronFleet | ✅ | ❌ | 😐 | 😐 |
| Ivy, mypyvy | 😐 | ✅ | 😐 | ✅ |
| Veil | ✅ | ✅ | ✅ | ✅ |

# Veil

| | |
|---|---|
| Arbitrary Properties | ✅ A shallowly-embedded DSL in Lean |
| Fast Feedback | ✅ Explicit-State and Symbolic Model Checking via SMT |
| Interactive Proofs for Complex (HO) Properties | ✅ Out-of-the-box interactive proofs in Lean |
| FOL Proof Automation | ✅ Uses external SMT solvers for proof goals in FOL |

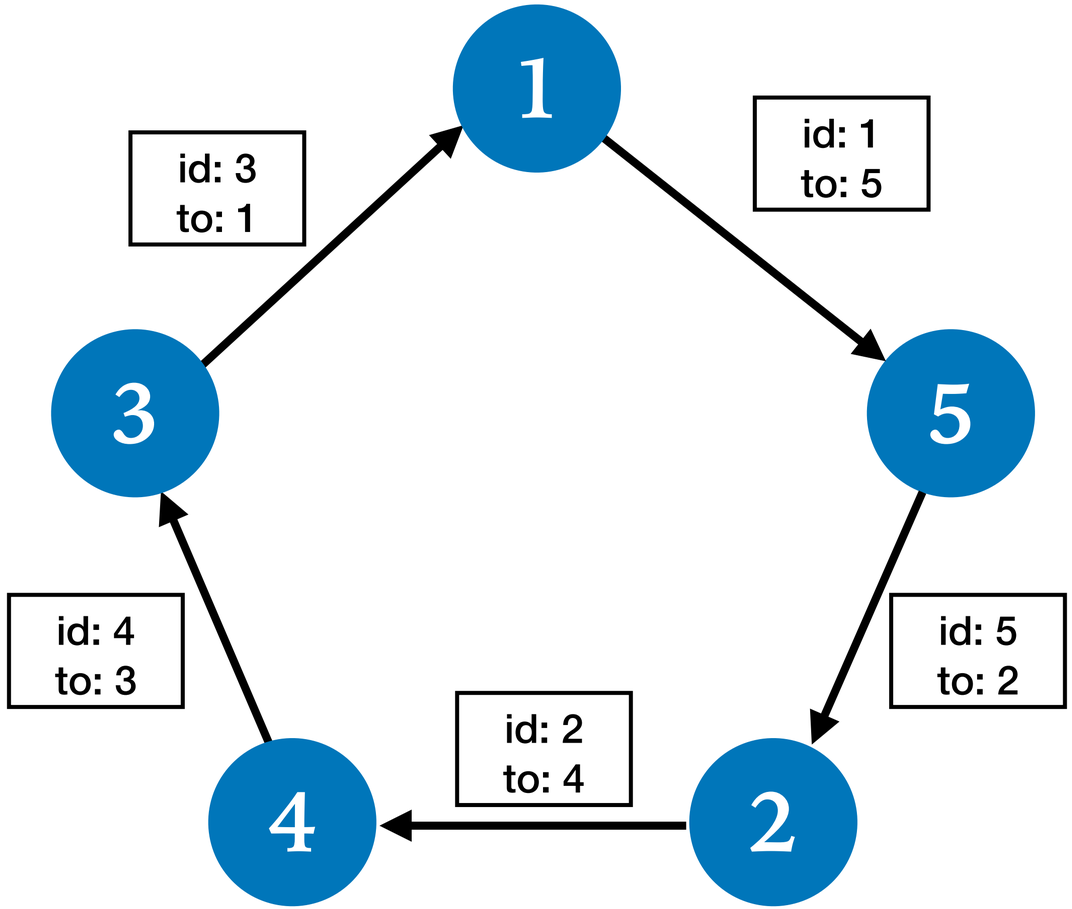
# The Rest of this Talk

- Veil in Action
- Details of the Implementation
- Case Studies

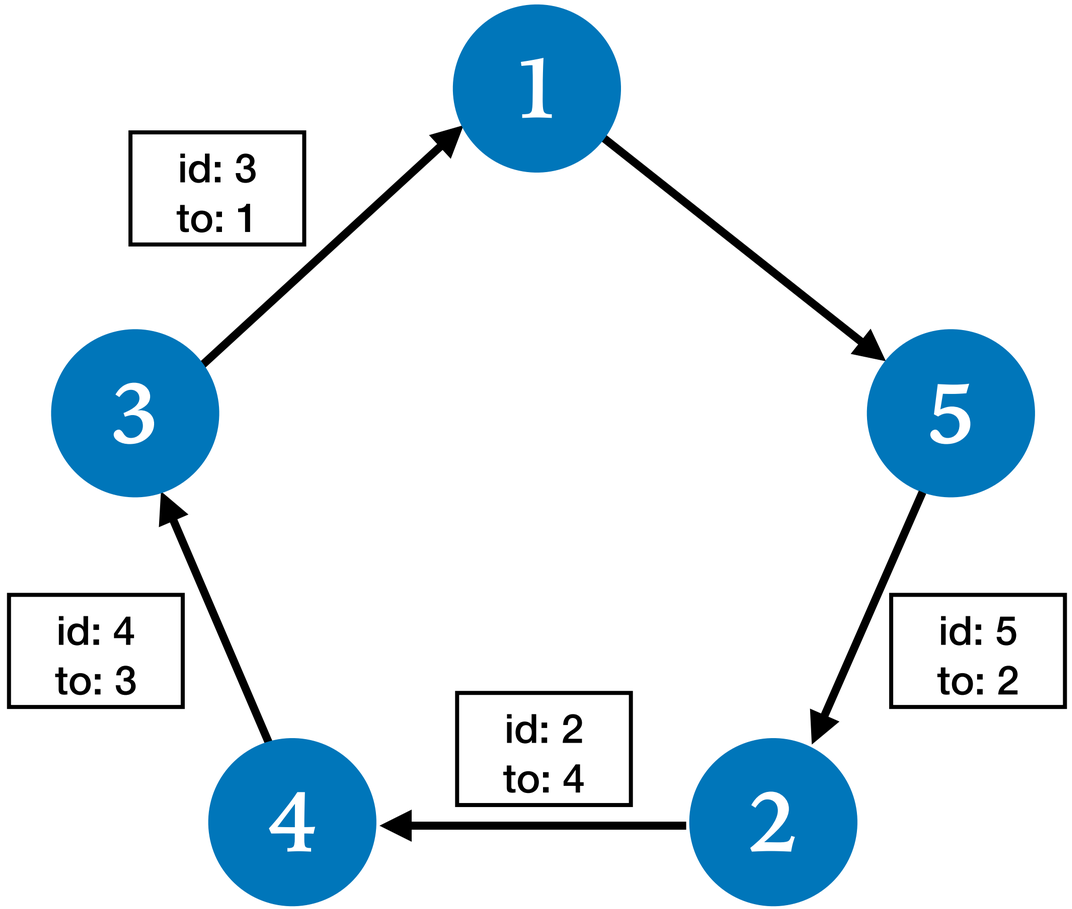
# The Rest of this Talk

- Veil in Action
- Details of the Implementation
- Case Studies

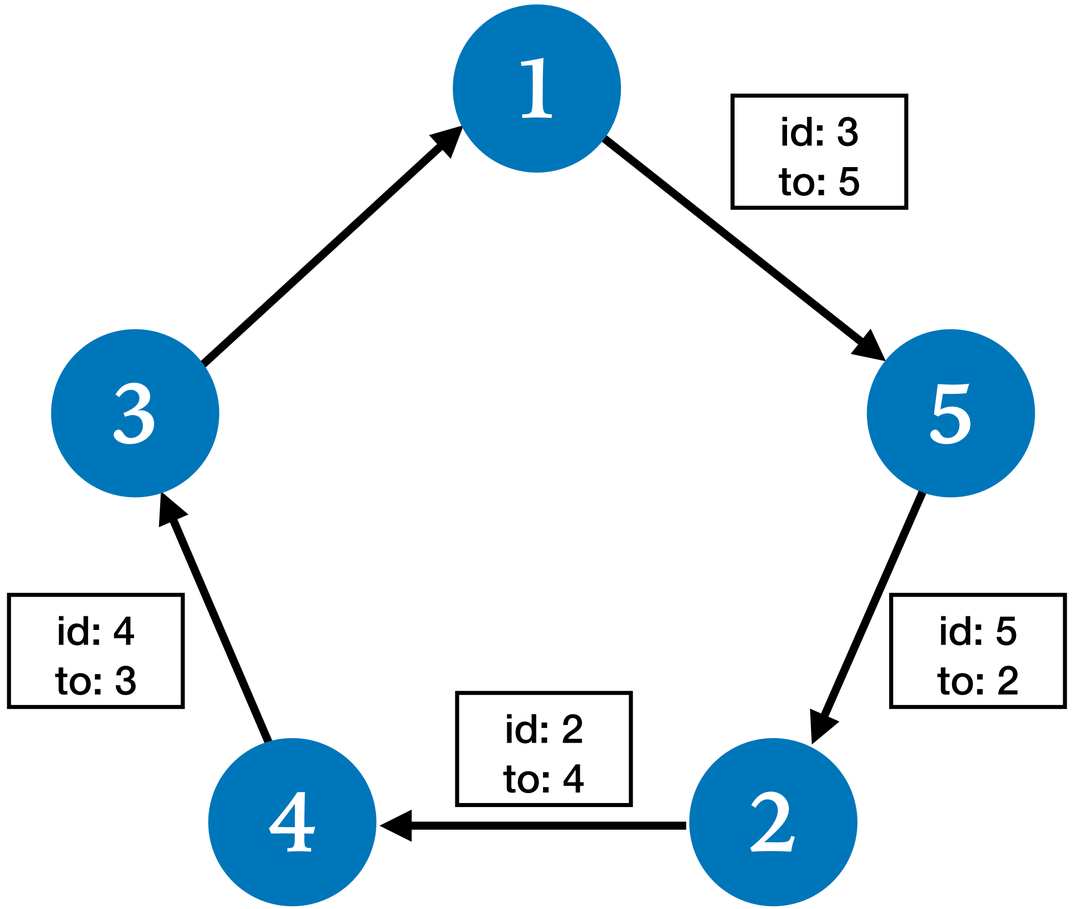
# Ring Leader Election



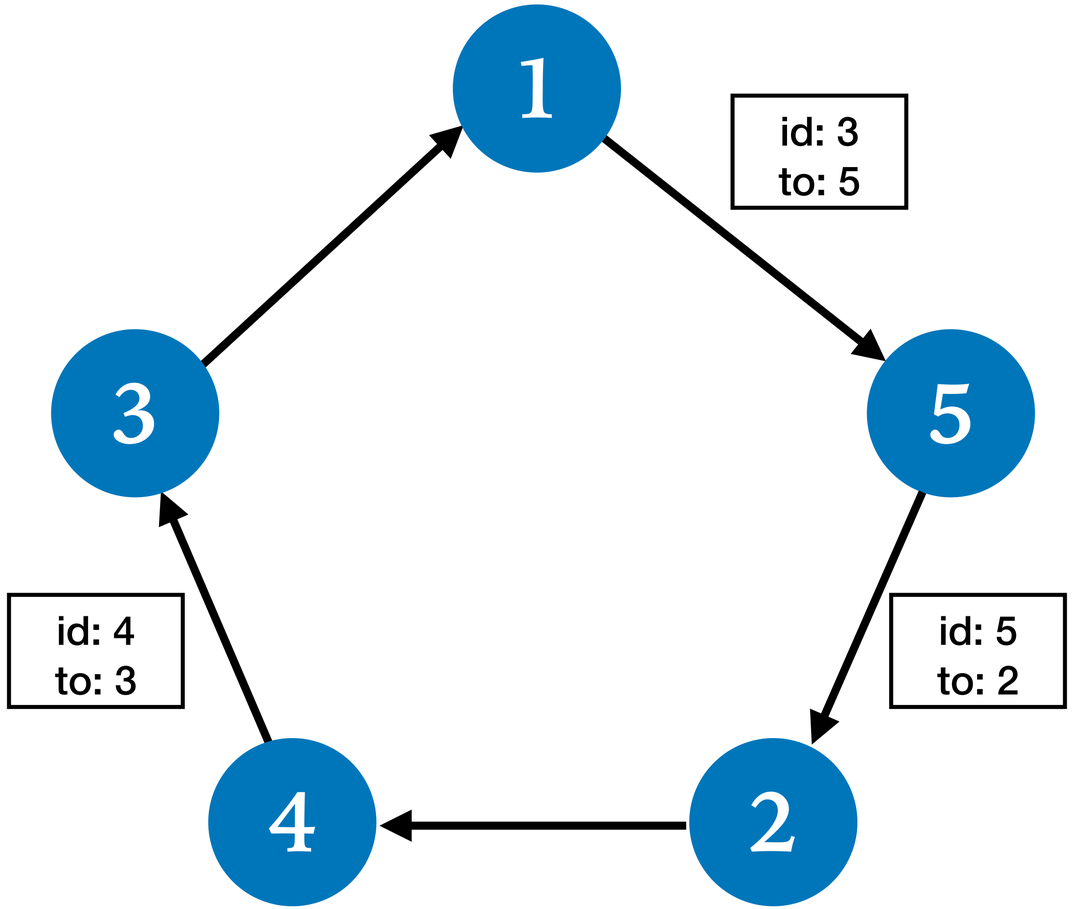
# Ring Leader Election



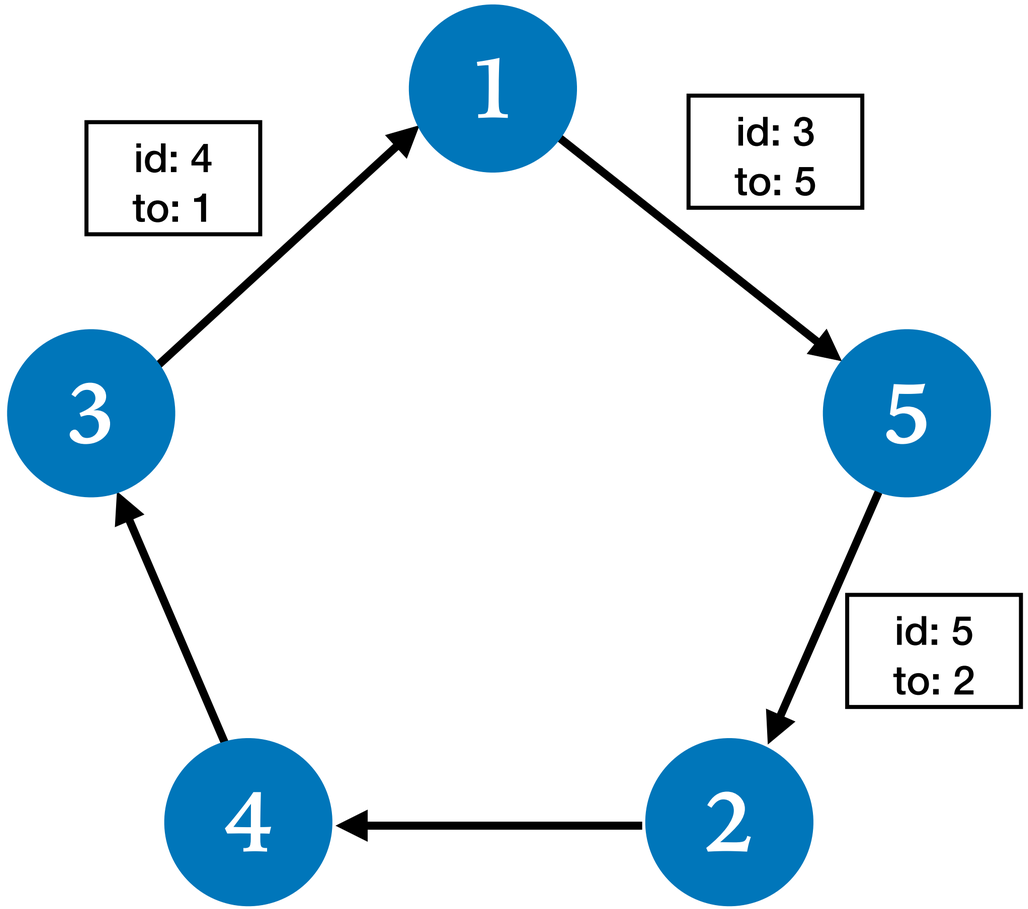
# Ring Leader Election



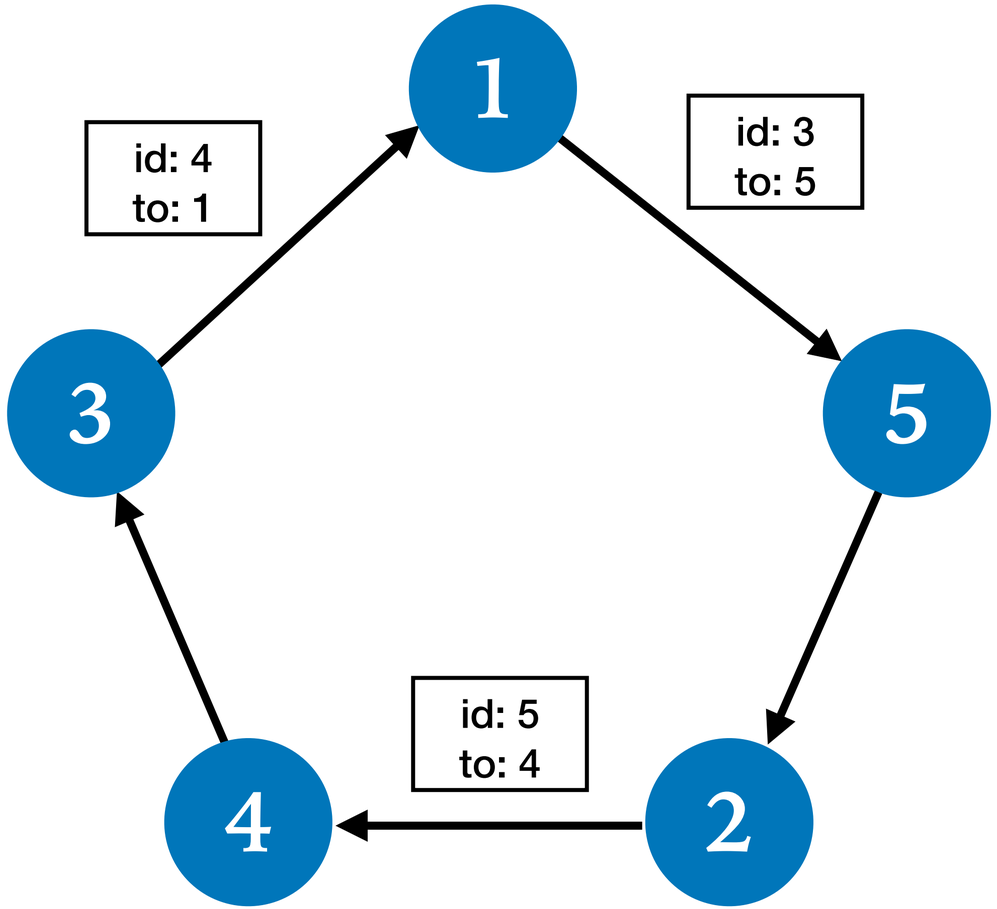
# Ring Leader Election



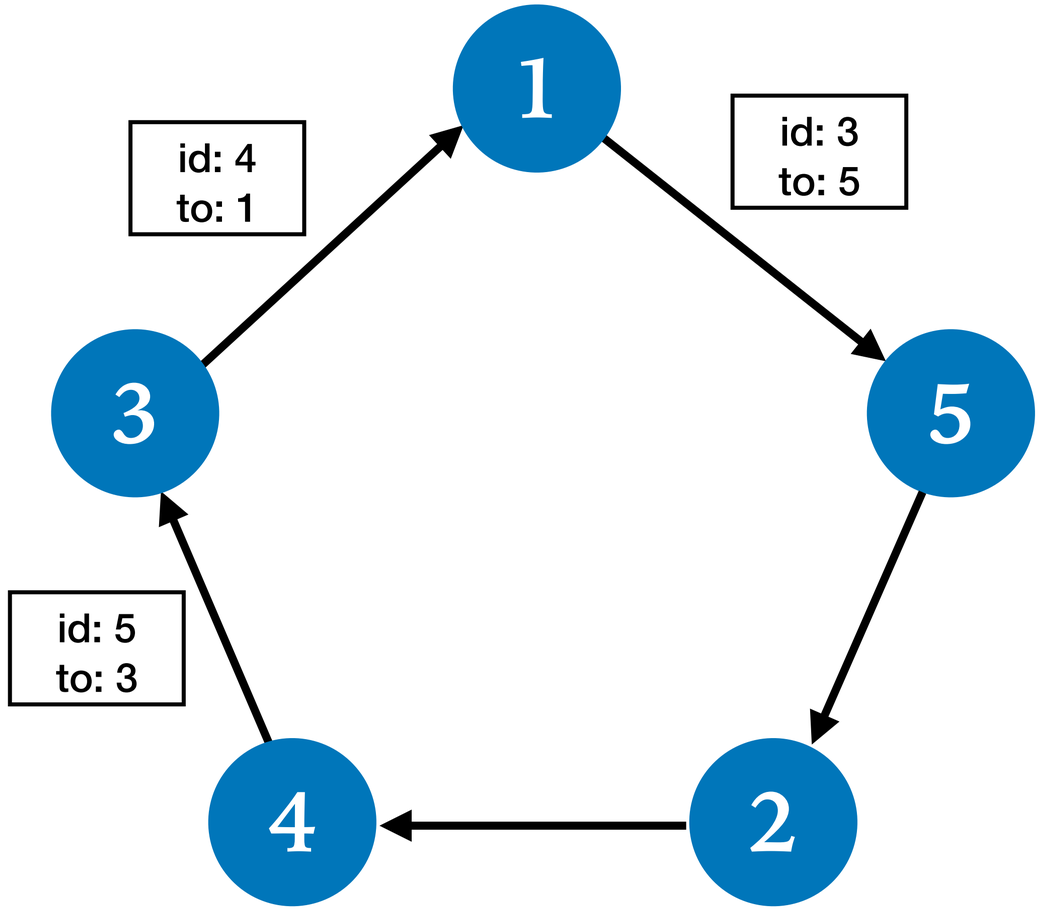
# Ring Leader Election



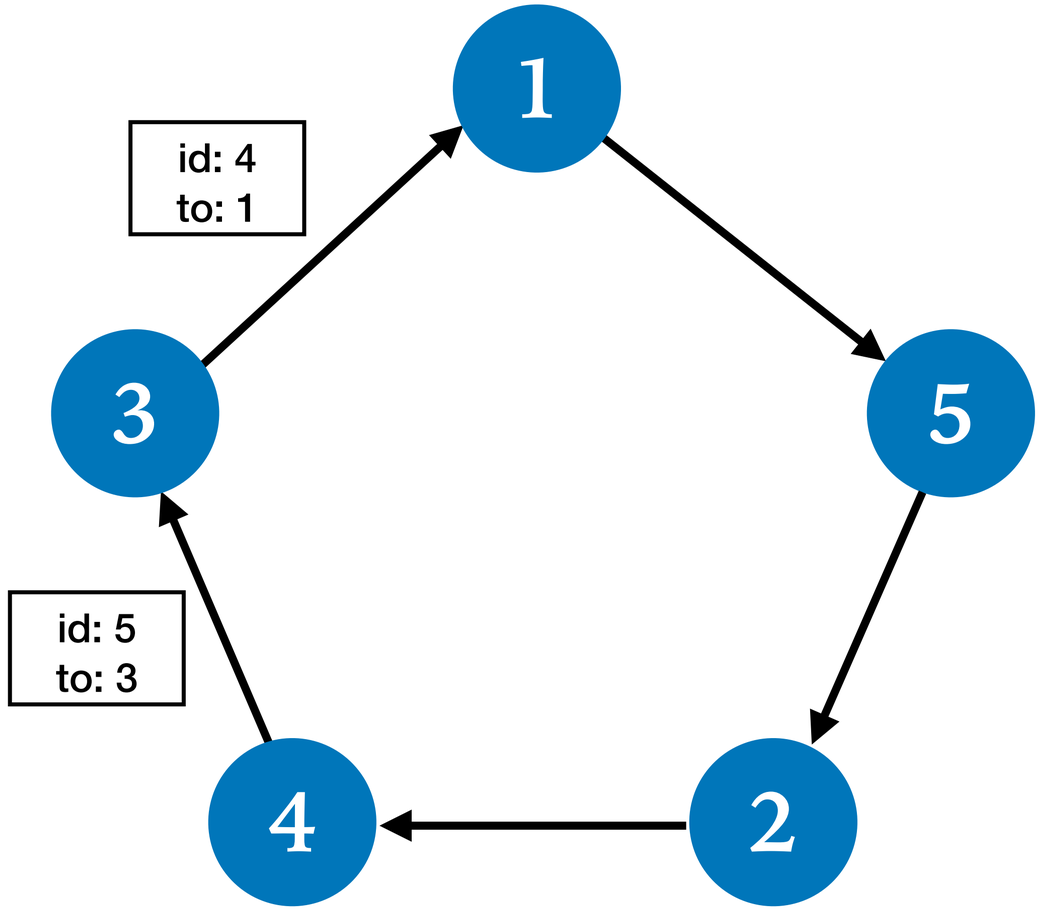
# Ring Leader Election



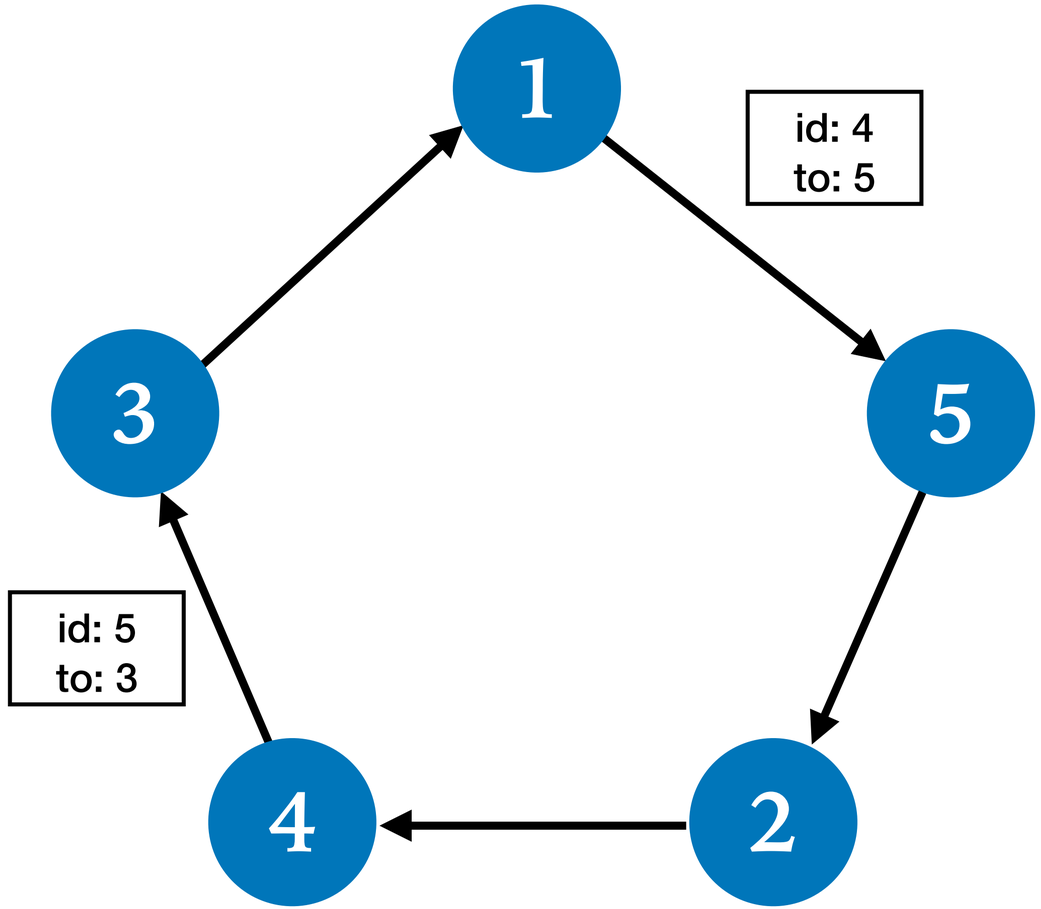
# Ring Leader Election



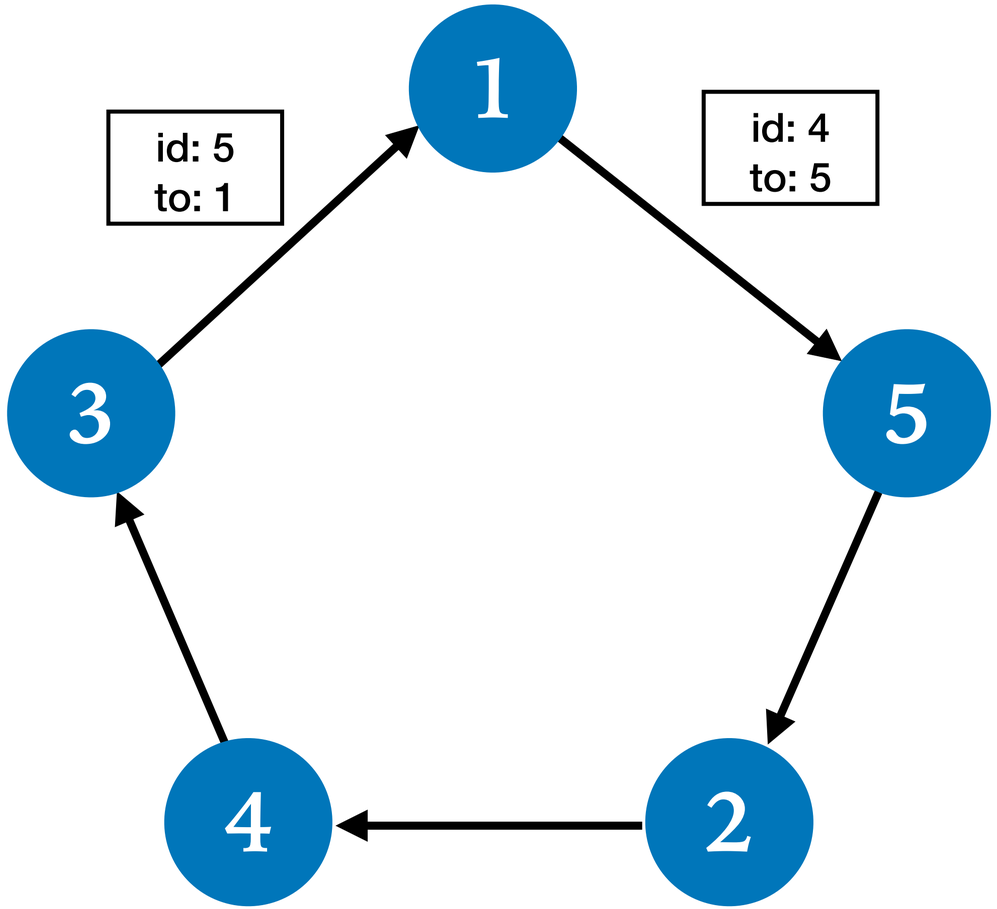
# Ring Leader Election



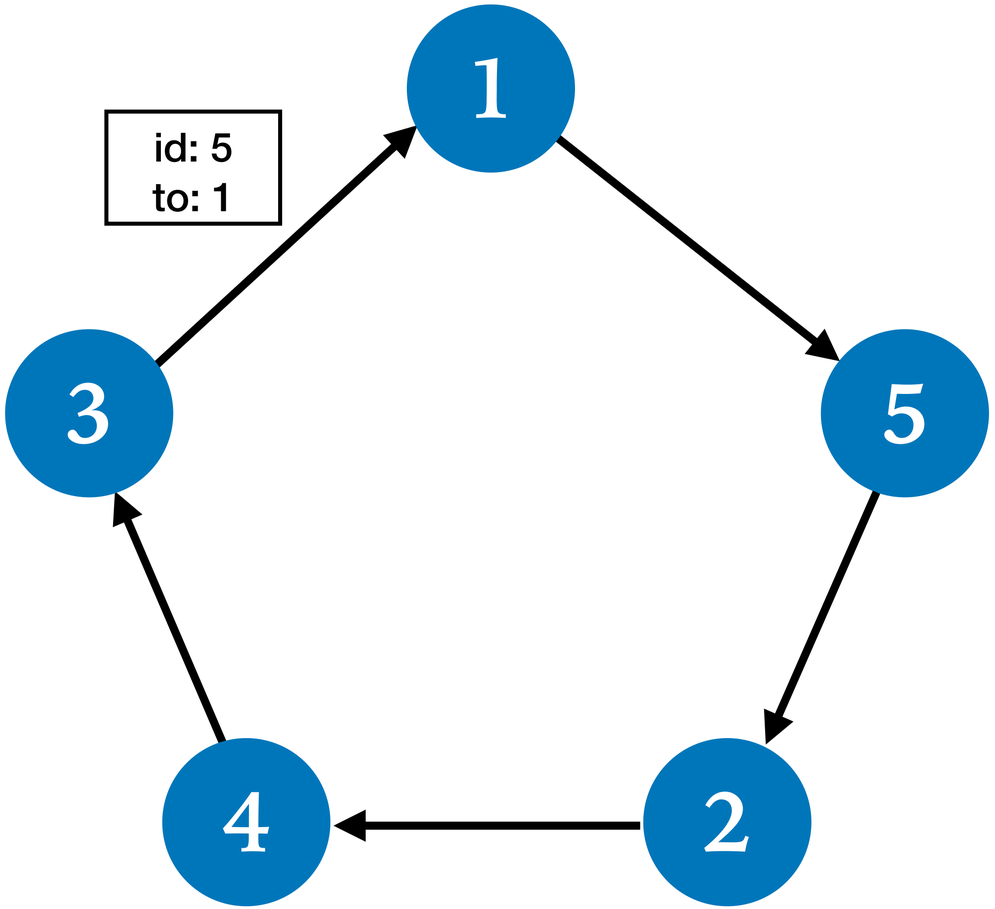
# Ring Leader Election



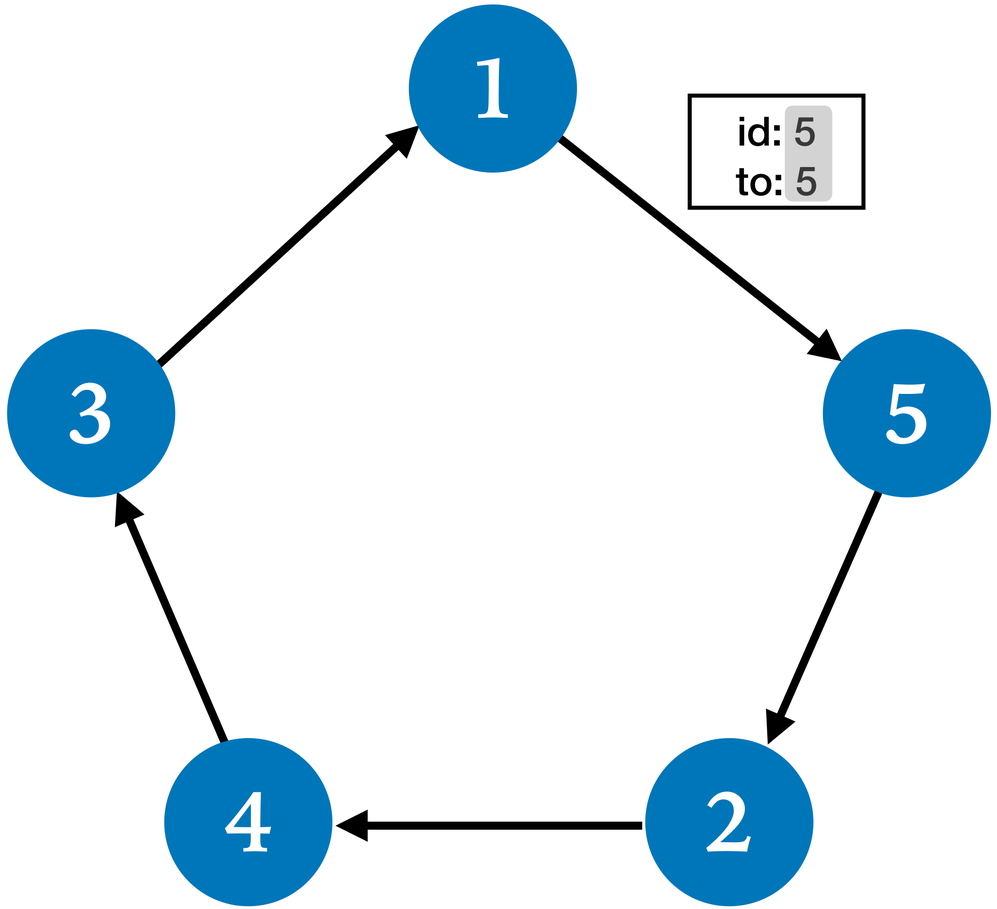
# Ring Leader Election



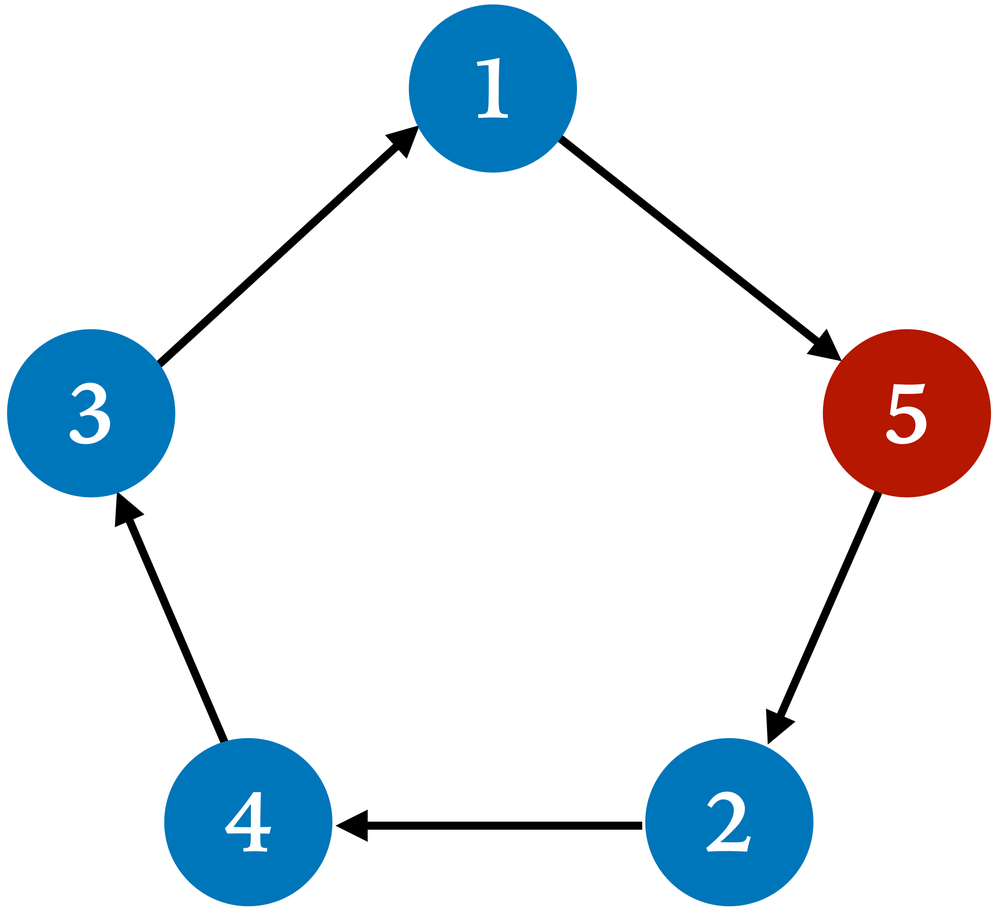
# Ring Leader Election



# Ring Leader Election

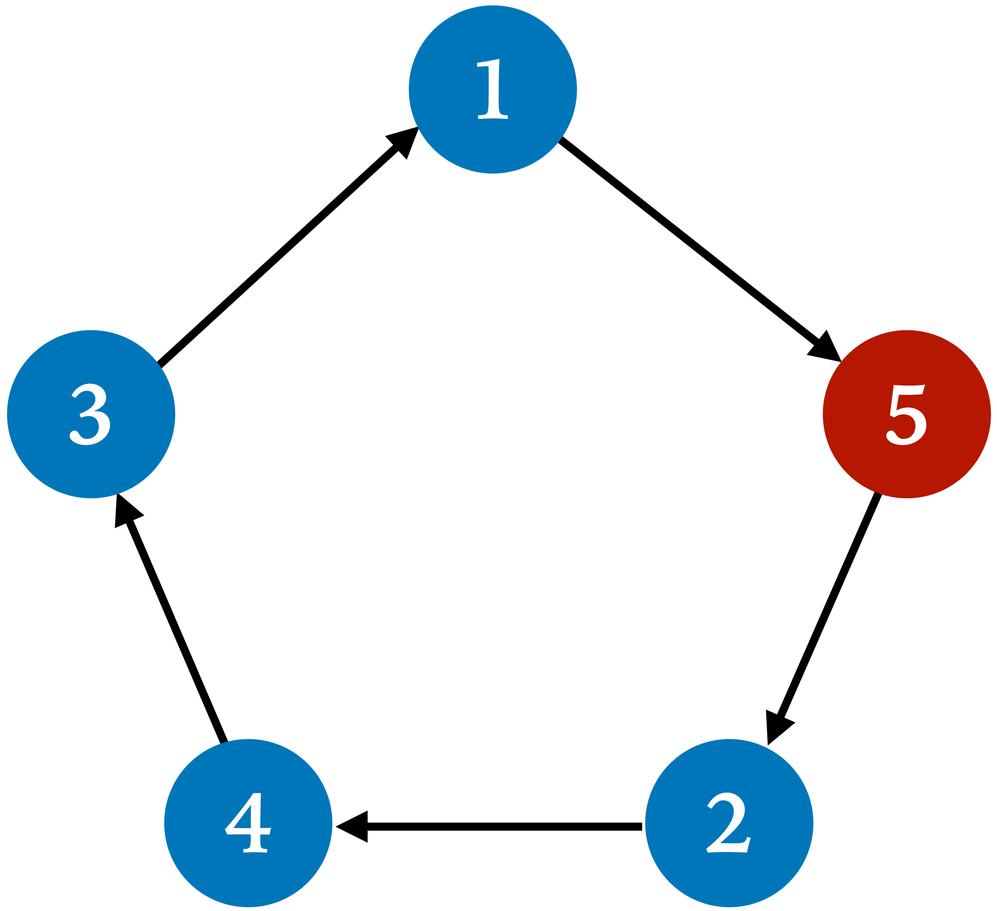


# Ring Leader Election




leader elected

# Safety Property to Verify




*There is at most one leader.*

# Live Demo

try.veil.dev

live.veil.dev

https://github.com/verse-lab/veil

branch “veil-2.0”

# Summary of the Demo

- Veil state: concrete and uninterpreted constrained types
- Interpretations are given for TLC-style explicit model checking
- Incomplete proofs can be done in the interactive proof mode
- First-Order encodings: BMC and deductive verification via SMT

# Talk Outline

- Veil in Action
- Details of the Implementation
- Case Studies

# Talk Outline

- Veil in Action
- Details of the Implementation
- Case Studies

# Generating Verification Conditions

# System Specification

initial state

```
after_init {
  leader N := False
  pending M N := False
}
```

actions

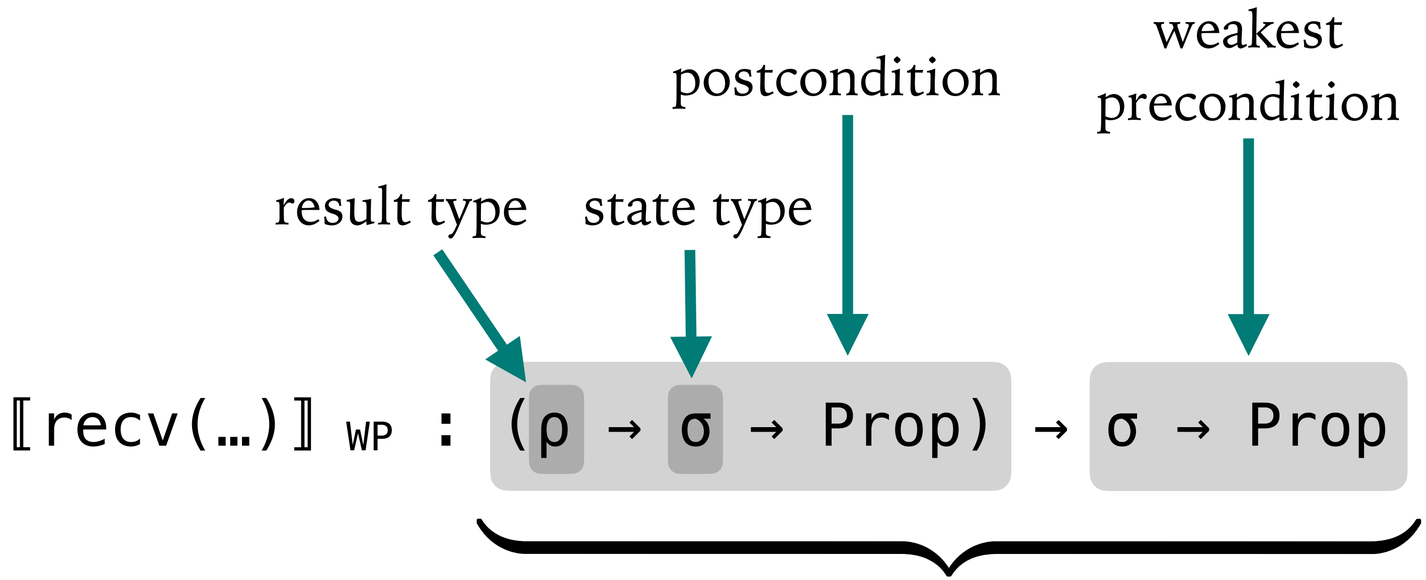
```
action send (n next : node) = {
  require n ≠ next ∧ ∀ Z,
          ((Z ≠ n ∧ Z ≠ next) → btw n next Z)
  pending n next := True
}

action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
}
```

safety properties

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2

invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

# Semantics of Actions

```
action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
}
```



# Verification Conditions for Safety

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2
```

```
invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

```
⟦act⟧ WP : (ρ → σ → Prop) → σ → Prop
```

```
after_init {
  leader N := False
  pending M N := False
}
```

```
Inv = λ (r: ρ) (s: σ). Inv s
```

1. ∀ s. ⟦after_init⟧ WP **Inv** s
2. ∀ act s. **Inv** s ⇒ ⟦act⟧ WP **Inv** s
3. ∀ s. Inv s ⇒ Safety s

# Generating Weakest Preconditions Automatically

$$[\![\texttt{act}]\!]_{\texttt{WP}} : \underbrace{(\rho \to \sigma \to \texttt{Prop}) \to \sigma \to \texttt{Prop}}$$

weakest precondition transformer (WP σ ρ)

```
def WP.pure (r : ρ) : WP σ ρ := fun s post => post r s

def WP.bind (c : WP σ ρ) (next : ρ → WP σ ρ) : WP σ ρ :=
                        fun s post => c (fun r s' => next r post s') s
```

*a.k.a.* a Dijkstra monad

*Verifying higher-order programs with the Dijkstra monad*, Swamy et al., PLDI'13

# Generating Weakest Preconditions Automatically

```
def WP.pure (r : ρ) : WP σ ρ := fun s post => post r s

def WP.bind (c : WP σ ρ) (next : ρ → WP σ ρ) : WP σ ρ :=
                                fun s post => c (fun r s' => next r post s') s
```

```
action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
}
```

# Generating Weakest Preconditions Automatically

```
def WP.pure (r : ρ) : WP σ ρ := fun s post => post r s

def WP.bind (c : WP σ ρ) (next : ρ → WP σ ρ) : WP σ ρ :=
  fun s post => c (fun r s' => next r post s') s
```

```
action recv (id n next : node) = do
  require isNext n next
  require pending id n

  let newPending ← fresh
  pending := newPending

  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
```

```
def fresh (τ : Type) : WP σ τ := fun s post => ∀ (t : τ), post t s
```

# Weakest Preconditions and Counterexamples

# Debugging a Specification

```
action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
}
```

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2

invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

# Debugging a Specification

```
action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le id n) then
      pending id next := True
}
```

```
#gen_spec

set_option veil.printCounterexamples true
set_option veil.smt.model.minimize true
set_option veil.vc_gen "wp"
#check_invariants
```

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2

invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

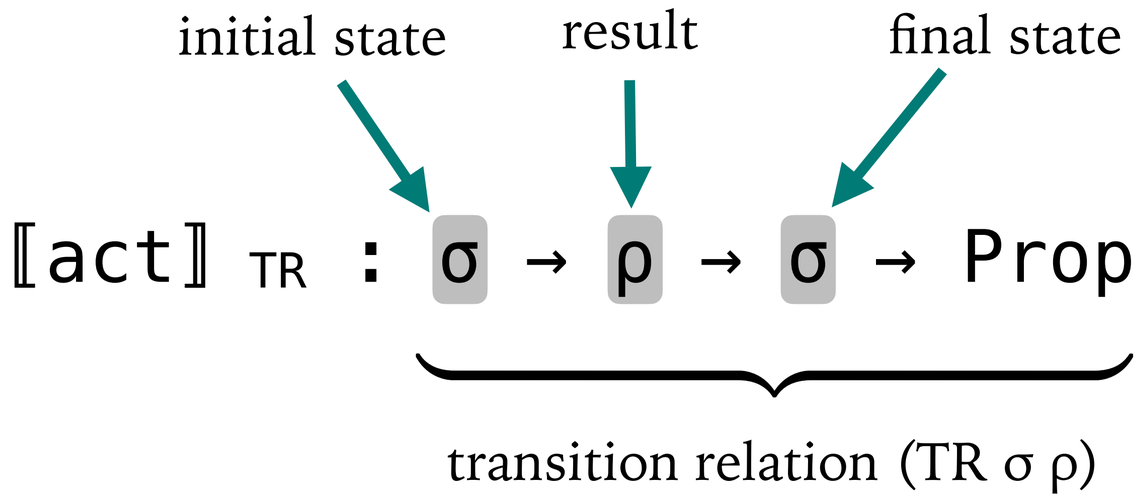
```
The following set of actions must preserve the invariant:
  …
  recv
    single_leader ... ✅
    leader_greatest ... ✅
    self_msg_only_if_greatest ... ❌
    no_bypass ... ❌

Counter-examples
================

self_msg_only_if_greatest:
sort node = #[node0, node1]
id = node0
n = node1
next = node0
st.pending(node0, node1) = true
tot.le(node0, node0) = true
tot.le(node0, node1) = true
tot.le(node1, node1) = true
```

Can we get a post-state (`st'`) too?

# Weakest Precondition Semantics

$$⟦\mathtt{act}⟧_{\mathtt{WP}} : (\rho \to \sigma \to \mathtt{Prop}) \to \sigma \to \mathtt{Prop}$$

# Two-State Transition Semantics



# Verification with Transition Semantics

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2

invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

```
⟦act⟧ TR : σ → ρ → σ → Prop

Inv = λ (r: ρ) (s: σ). Inv s
```

1. ∀ $s_0$ s. ⟦after_init⟧ TR $s_0$ s ⇒ ***Inv*** s
2. ∀ act s r s'. ***Inv*** s ∧ ⟦act⟧ TR s r s' ⇒ ***Inv*** s'
3. ∀ s. Inv s ⇒ Safety s

# Debugging a Specification, Reloaded

```
action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le id n) then
      pending id next := True
}
```

```
#gen_spec

set_option veil.printCounterexamples true
set_option veil.smt.model.minimize true
set_option veil.vc_gen "transition"
#check_invariants
```

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2

invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

```
The following set of actions must preserve the invariant:
  …
  recv
    single_leader ... ✅
    leader_greatest ... ✅
    self_msg_only_if_greatest ... ❌
    no_bypass ... ❌

Counter-examples
================

self_msg_only_if_greatest:
sort node = #[node0, node1]
id = node1
n = node0
next = node1
st.pending(node1, node0) = true
st'.pending(node1, node1) = true
tot.le(node0, node0) = true
tot.le(node1, node0) = true
tot.le(node1, node1) = true
```

# Transition Semantics and Monadic Embedding

```
def TR.pure (r : ρ) : TR σ ρ := fun s₁ r' s₂ => s₁ = s₂ ∧ r' = r

def TR.bind (c : TR σ ρ) (next : ρ → TR σ ρ') : TR σ ρ' :=
    fun s₁ r' s₂ => ∃ r s', (c s₁ r s') ∧ (next r s' r' s₂)
```

HO quantification (because `s'` contains relations):

**VCs cannot be discharged via SMT**

# Alternative Definition of Transition Semantics

postcondition that *excludes*
the result $r_1$ and state $s_1$

$$⟦\mathtt{act}⟧_{TR'} : TR\ \sigma\ \rho \triangleq \lambda\ s_0\ r_1\ s_1.\ \neg\ ⟦\mathtt{act}⟧_{WP}\ (\lambda\ r\ s.\ \neg(r = r_1 \wedge s = s_1))\ s_0$$

the result $r_1$ and state $s_1$
are *not unreachable* from $s_0$

# Adequacy Theorem

```
∀ act s post, ⟦act⟧ WP post s ⇔
              (∀ s' r, ⟦act⟧ TR' s r s' ⇒ post r s')
```

"A (pre-)state `s` satisfies `⟦act⟧ WP post`
if and only if
*any* final state `s'` reachable from `s` via `⟦act⟧ TR'` satisfies `post`."

# Summary of the Implementation

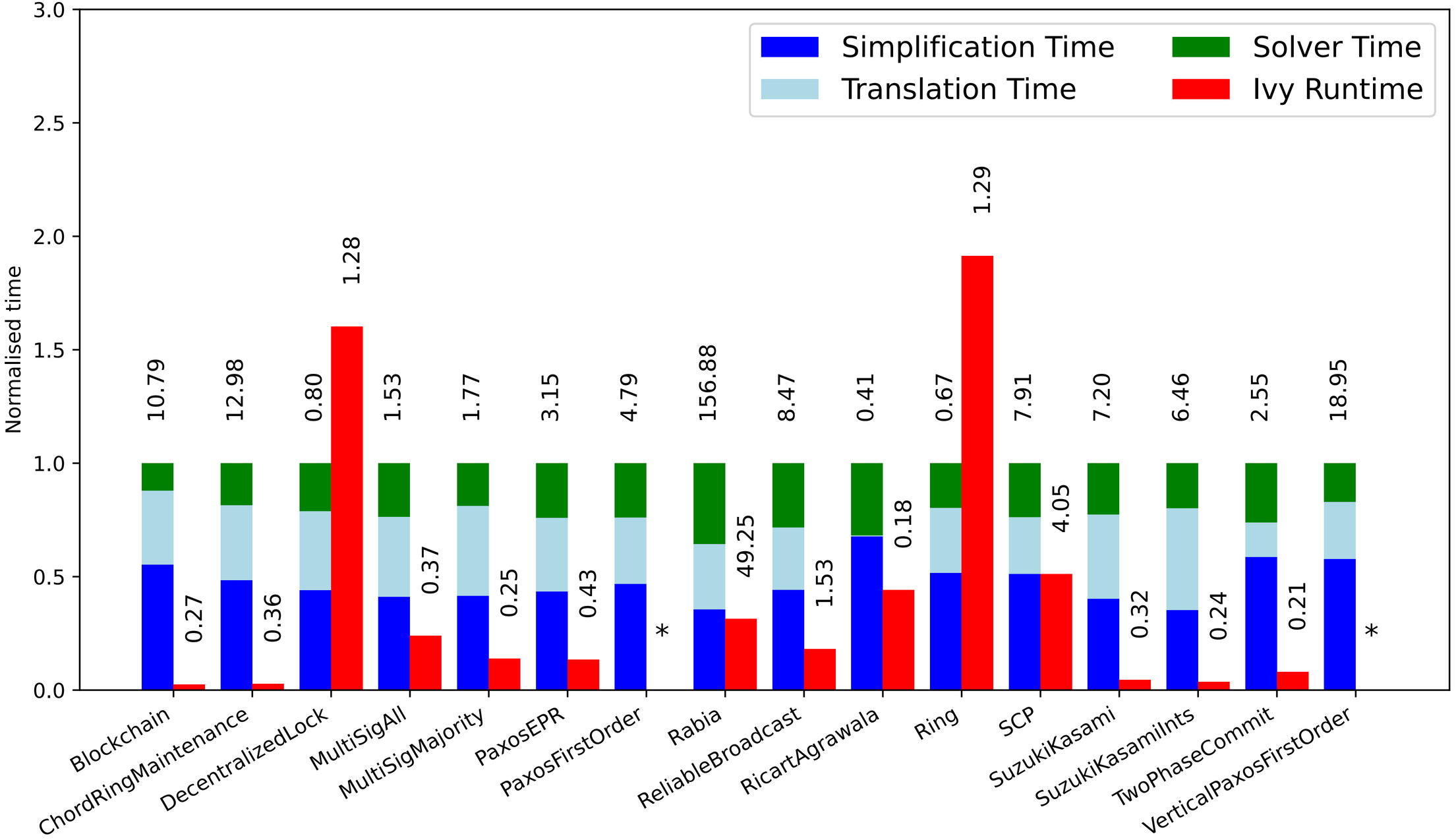
- The **Veil** language is shallowly embedded in Lean using *Dijkstra monads*
- Allows for *different VC generators* (currently, WP and Two-State Transition)
- Adequacy of VC generators is *proven mechanically as a theorem*
- Concrete and symbolic execution via Lean's *type class instance* mechanism

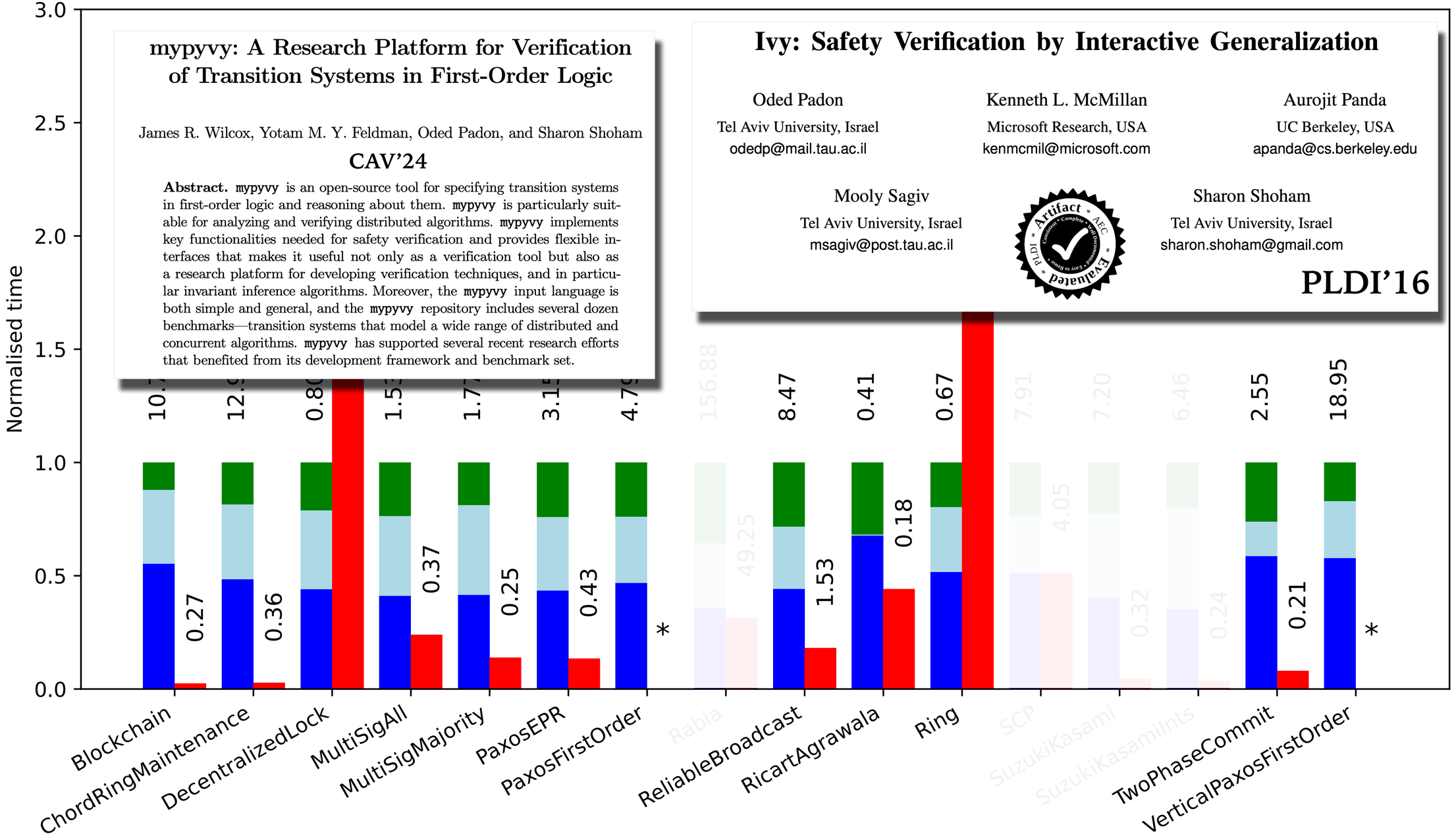
# Talk Outline

- Veil in Action
- Details of the Implementation
- Case Studies

# Talk Outline

- Veil in Action
- Details of the Implementation
- Case Studies


Simplification Time
Translation Time
Solver Time
Ivy Runtime
Normalised time
3.0
2.5
2.0
1.5
1.0
0.5
0.0
10.79
0.27
12.98
0.36
0.80
1.28
1.53
0.37
1.77
0.25
3.15
0.43
4.79
*
156.88
49.25
8.47
1.53
0.41
0.18
0.67
1.29
7.91
4.05
7.20
0.32
6.46
0.24
2.55
0.21
18.95
*
Blockchain
ChordRingMaintenance
DecentralizedLock
MultiSigAll
MultiSigMajority
PaxosEPR
PaxosFirstOrder
Rabia
ReliableBroadcast
RicartAgrawala
Ring
SCP
SuzukiKasami
SuzukiKasamiInts
TwoPhaseCommit
VerticalPaxosFirstOrder




**mypyvy: A Research Platform for Verification of Transition Systems in First-Order Logic**

James R. Wilcox, Yotam M. Y. Feldman, Oded Padon, and Sharon Shoham

**CAV'24**

**Abstract.** `mypyvy` is an open-source tool for specifying transition systems in first-order logic and reasoning about them. `mypyvy` is particularly suitable for analyzing and verifying distributed algorithms. `mypyvy` implements key functionalities needed for safety verification and provides flexible interfaces that makes it useful not only as a verification tool but also as a research platform for developing verification techniques, and in particular invariant inference algorithms. Moreover, the `mypyvy` input language is both simple and general, and the `mypyvy` repository includes several dozen benchmarks—transition systems that model a wide range of distributed and concurrent algorithms. `mypyvy` has supported several recent research efforts that benefited from its development framework and benchmark set.

**Ivy: Safety Verification by Interactive Generalization**

Oded Padon
Tel Aviv University, Israel
odedp@mail.tau.ac.il

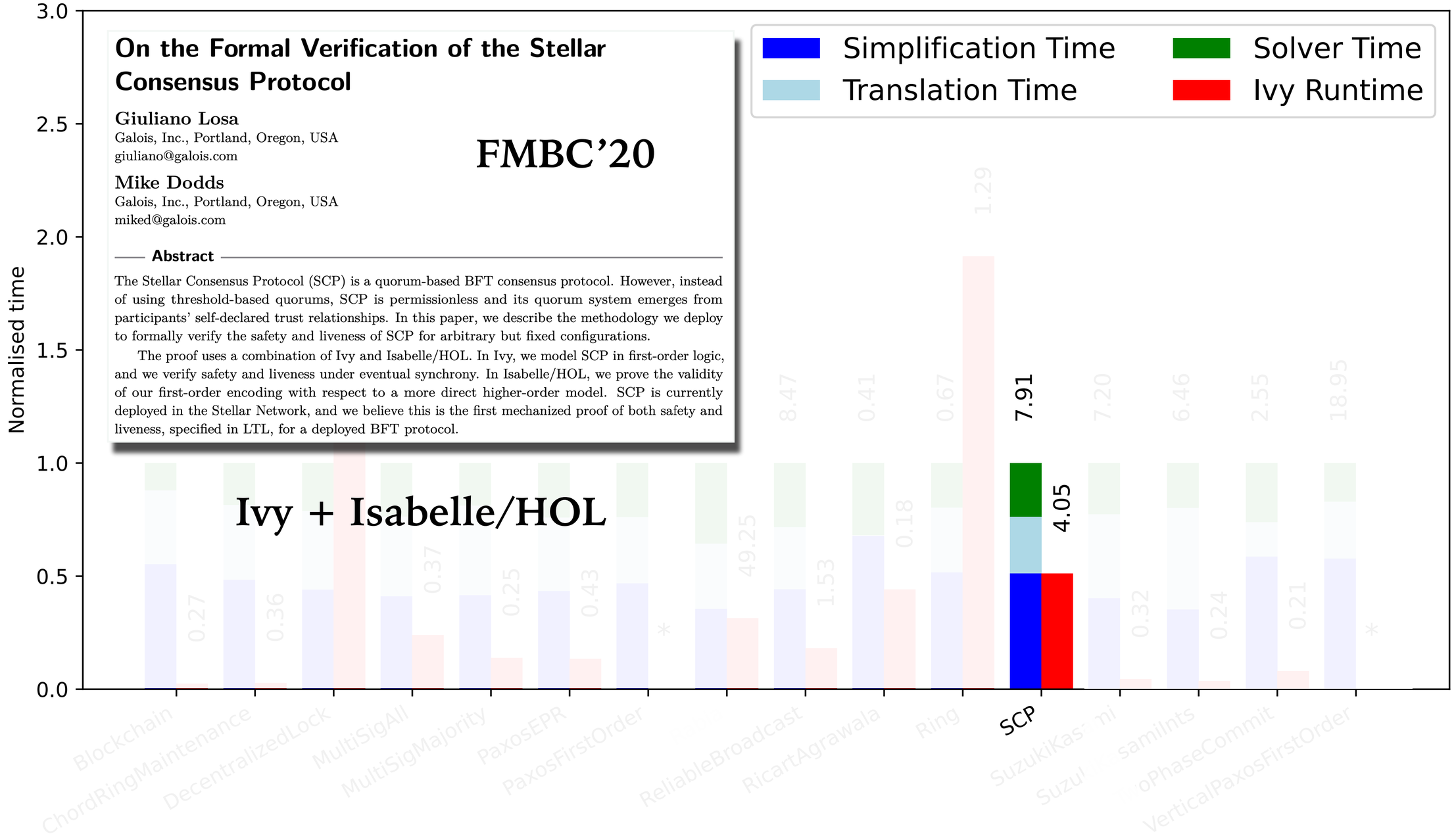
Kenneth L. McMillan
Microsoft Research, USA
kenmcmil@microsoft.com

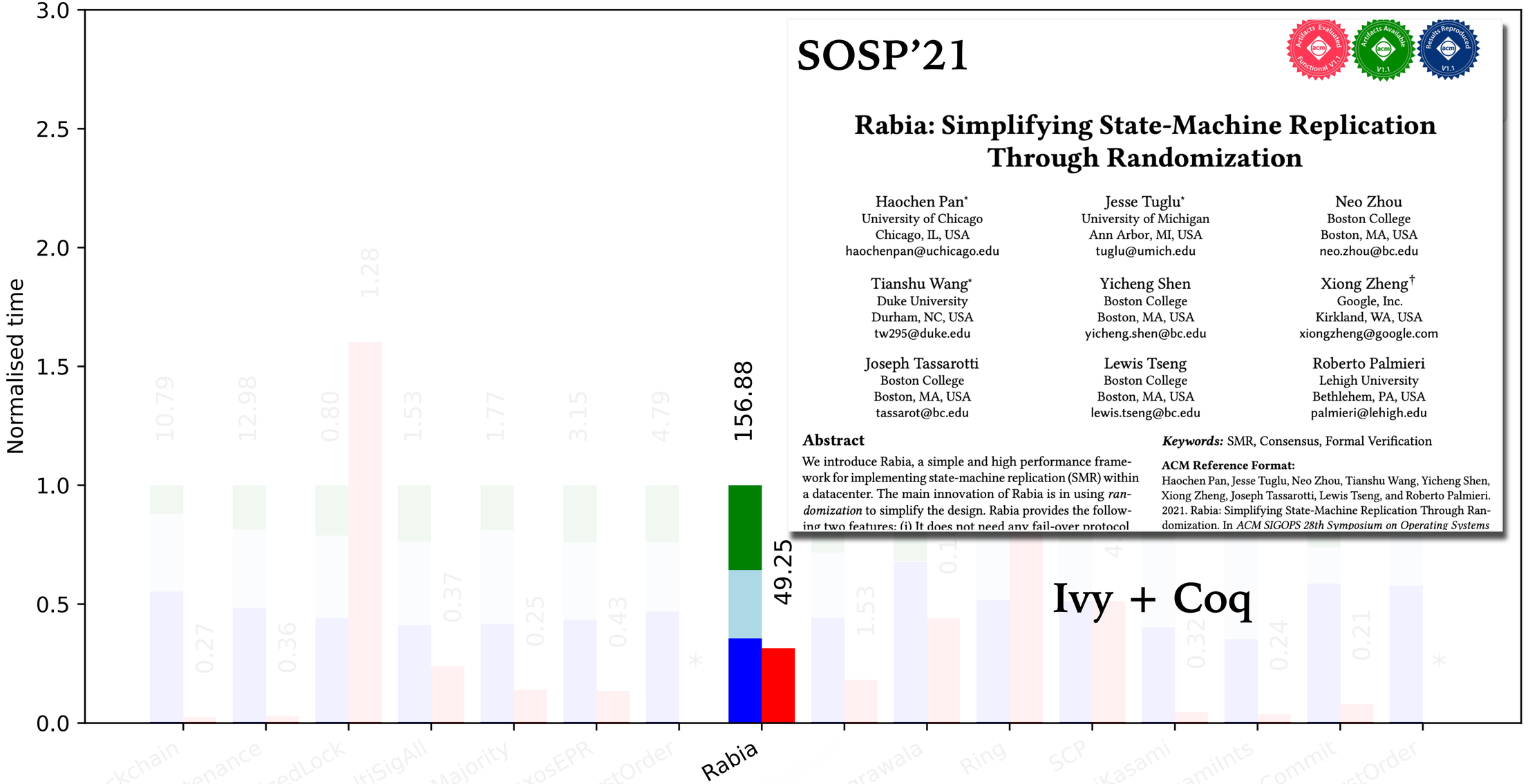
Aurojit Panda
UC Berkeley, USA
apanda@cs.berkeley.edu

Mooly Sagiv
Tel Aviv University, Israel
msagiv@post.tau.ac.il

Sharon Shoham
Tel Aviv University, Israel
sharon.shoham@gmail.com

**PLDI'16**


On the Formal Verification of the Stellar Consensus Protocol
Giuliano Losa
Galois, Inc., Portland, Oregon, USA
giuliano@galois.com
Mike Dodds
Galois, Inc., Portland, Oregon, USA
miked@galois.com
FMBC'20
Abstract
The Stellar Consensus Protocol (SCP) is a quorum-based BFT consensus protocol. However, instead of using threshold-based quorums, SCP is permissionless and its quorum system emerges from participants' self-declared trust relationships. In this paper, we describe the methodology we deploy to formally verify the safety and liveness of SCP for arbitrary but fixed configurations.
The proof uses a combination of Ivy and Isabelle/HOL. In Ivy, we model SCP in first-order logic, and we verify safety and liveness under eventual synchrony. In Isabelle/HOL, we prove the validity of our first-order encoding with respect to a more direct higher-order model. SCP is currently deployed in the Stellar Network, and we believe this is the first mechanized proof of both safety and liveness, specified in LTL, for a deployed BFT protocol.
Ivy + Isabelle/HOL
Simplification Time
Translation Time
Solver Time
Ivy Runtime
Normalised time
3.0
2.5
2.0
1.5
1.0
0.5
0.0
7.91
4.05
SCP
Blockchain
ChordRingMaintenance
DecentralizedLock
MultiSigAll
MultiSigMajority
PaxosEPR
PaxosFirstOrder
ReliableBroadcast
RicartAgrawala
Ring
SuzukiKas
Suzul
samiInts
oPhaseCommit
VerticalPaxosFirstOrder




SOSP'21

Artifacts Evaluated Functional V1.1 · Artifacts Available V1.1 · Results Reproduced V1.1

# Rabia: Simplifying State-Machine Replication Through Randomization


Haochen Pan*
University of Chicago
Chicago, IL, USA
haochenpan@uchicago.edu

Jesse Tuglu*
University of Michigan
Ann Arbor, MI, USA
tuglu@umich.edu

Neo Zhou
Boston College
Boston, MA, USA
neo.zhou@bc.edu

Tianshu Wang*
Duke University
Durham, NC, USA
tw295@duke.edu

Yicheng Shen
Boston College
Boston, MA, USA
yicheng.shen@bc.edu

Xiong Zheng†
Google, Inc.
Kirkland, WA, USA
xiongzheng@google.com

Joseph Tassarotti
Boston College
Boston, MA, USA
tassarot@bc.edu

Lewis Tseng
Boston College
Boston, MA, USA
lewis.tseng@bc.edu

Roberto Palmieri
Lehigh University
Bethlehem, PA, USA
palmieri@lehigh.edu



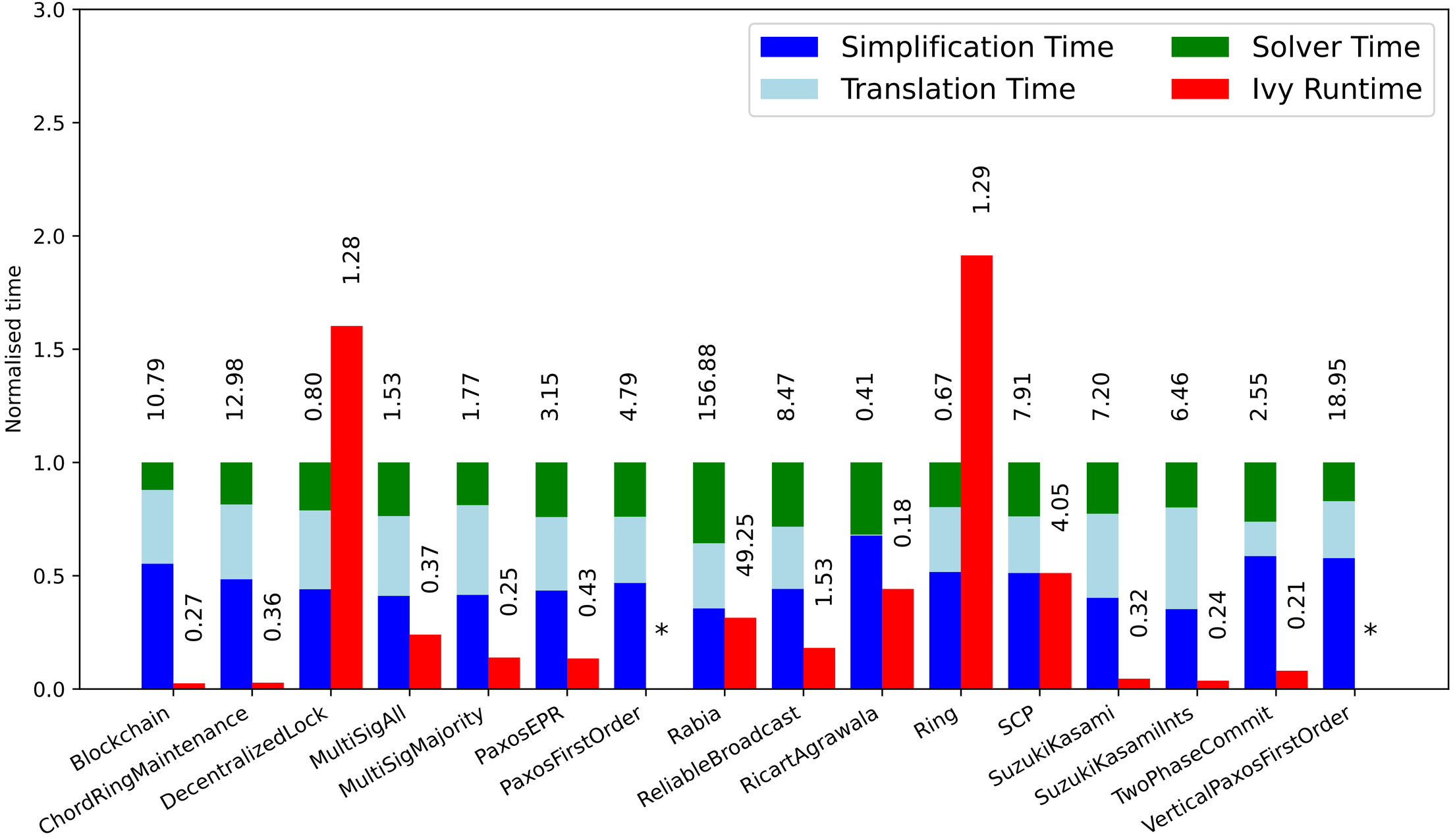
**Abstract**

We introduce Rabia, a simple and high performance framework for implementing state-machine replication (SMR) within a datacenter. The main innovation of Rabia is in using *randomization* to simplify the design. Rabia provides the following two features: (i) It does not need any fail-over protocol



***Keywords:*** SMR, Consensus, Formal Verification

**ACM Reference Format:**
Haochen Pan, Jesse Tuglu, Neo Zhou, Tianshu Wang, Yicheng Shen, Xiong Zheng, Joseph Tassarotti, Lewis Tseng, and Roberto Palmieri. 2021. Rabia: Simplifying State-Machine Replication Through Randomization. In *ACM SIGOPS 28th Symposium on Operating Systems*


Simplification Time
Translation Time
Solver Time
Ivy Runtime
Normalised time
3.0
2.5
2.0
1.5
1.0
0.5
0.0
10.79
0.27
12.98
0.36
0.80
1.28
1.53
0.37
1.77
0.25
3.15
0.43
4.79
*
156.88
49.25
8.47
1.53
0.41
0.18
0.67
1.29
7.91
4.05
7.20
0.32
6.46
0.24
2.55
0.21
18.95
*
Blockchain
ChordRingMaintenance
DecentralizedLock
MultiSigAll
MultiSigMajority
PaxosEPR
PaxosFirstOrder
Rabia
ReliableBroadcast
RicartAgrawala
Ring
SCP
SuzukiKasami
SuzukiKasamiInts
TwoPhaseCommit
VerticalPaxosFirstOrder

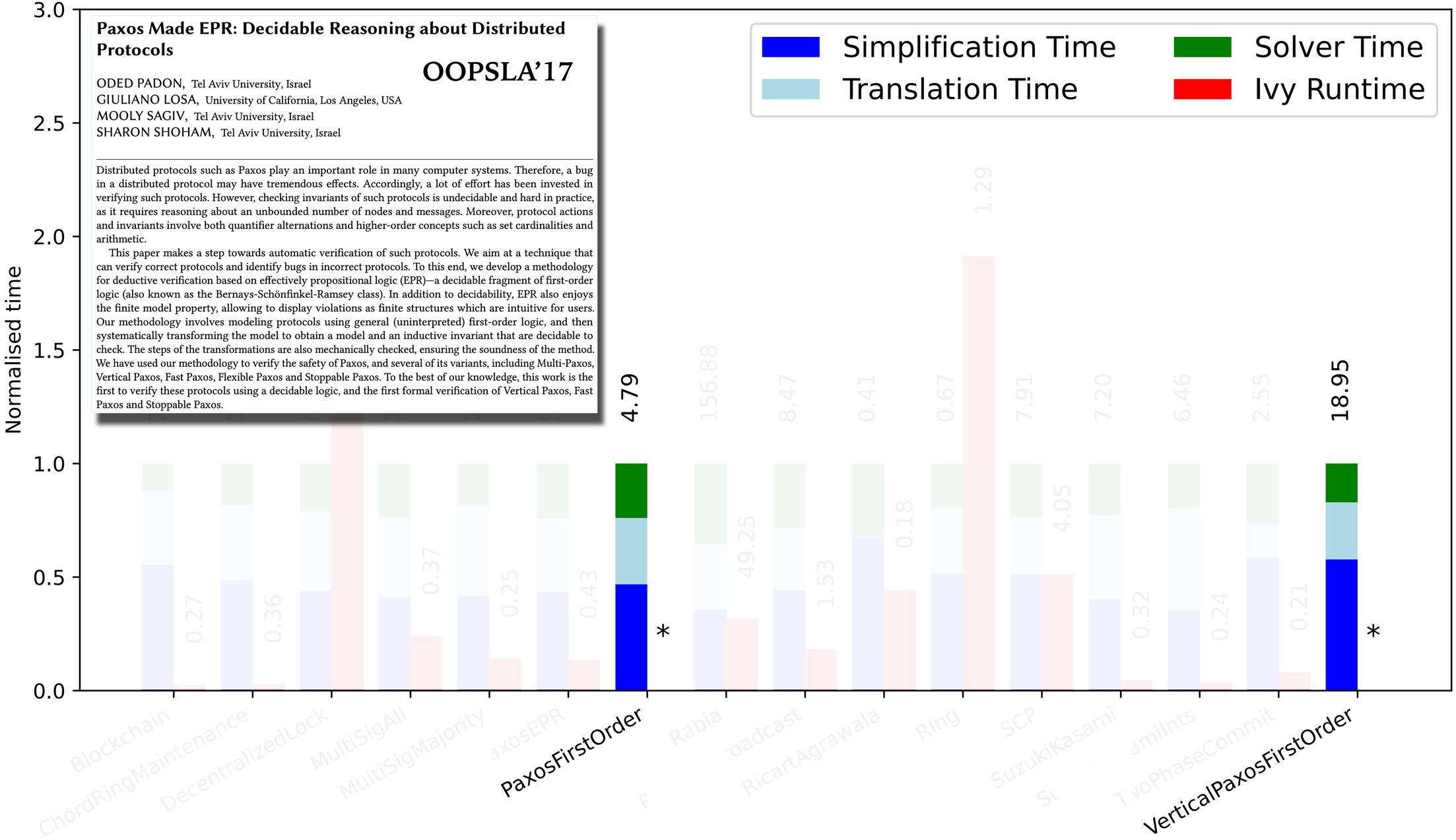
**Paxos Made EPR: Decidable Reasoning about Distributed Protocols**

**OOPSLA'17**

ODED PADON, Tel Aviv University, Israel
GIULIANO LOSA, University of California, Los Angeles, USA
MOOLY SAGIV, Tel Aviv University, Israel
SHARON SHOHAM, Tel Aviv University, Israel

Distributed protocols such as Paxos play an important role in many computer systems. Therefore, a bug in a distributed protocol may have tremendous effects. Accordingly, a lot of effort has been invested in verifying such protocols. However, checking invariants of such protocols is undecidable and hard in practice, as it requires reasoning about an unbounded number of nodes and messages. Moreover, protocol actions and invariants involve both quantifier alternations and higher-order concepts such as set cardinalities and arithmetic.

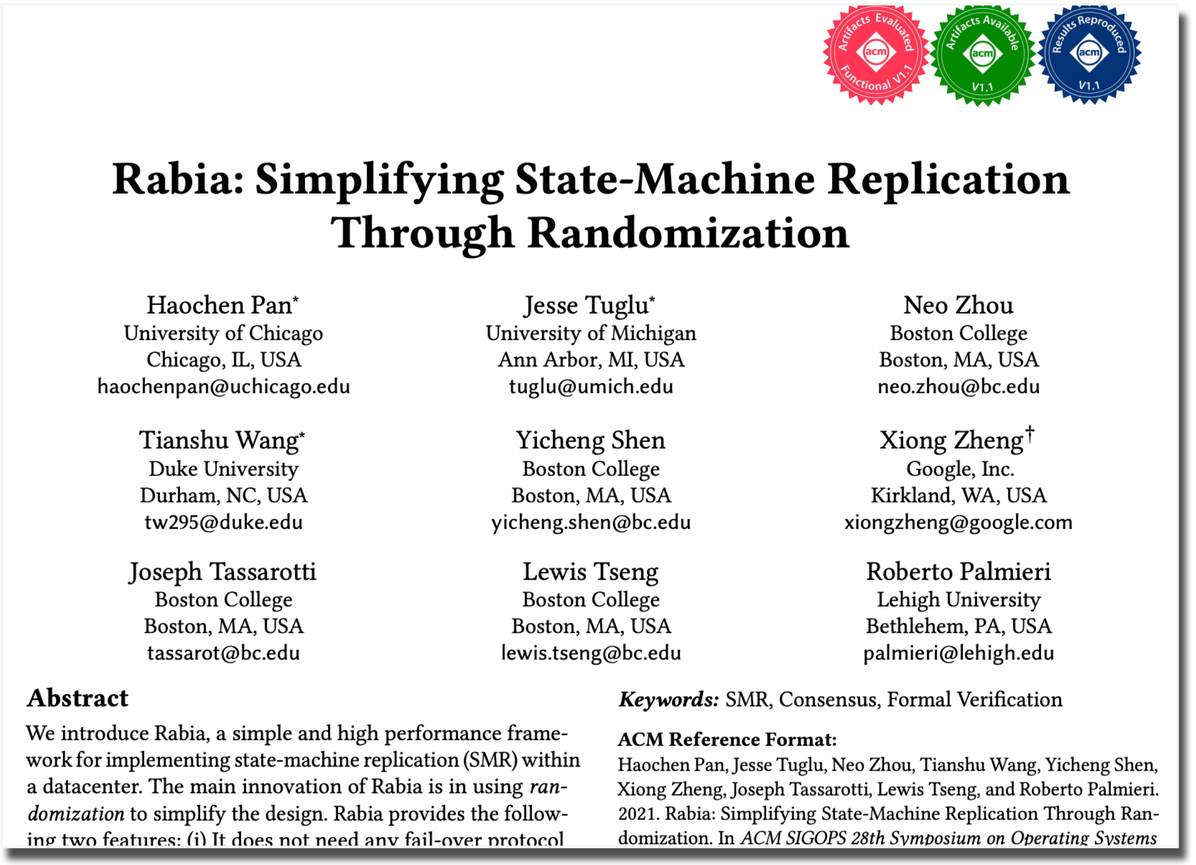
This paper makes a step towards automatic verification of such protocols. We aim at a technique that can verify correct protocols and identify bugs in incorrect protocols. To this end, we develop a methodology for deductive verification based on effectively propositional logic (EPR)—a decidable fragment of first-order logic (also known as the Bernays-Schönfinkel-Ramsey class). In addition to decidability, EPR also enjoys the finite model property, allowing to display violations as finite structures which are intuitive for users. Our methodology involves modeling protocols using general (uninterpreted) first-order logic, and then systematically transforming the model to obtain a model and an inductive invariant that are decidable to check. The steps of the transformations are also mechanically checked, ensuring the soundness of the method. We have used our methodology to verify the safety of Paxos, and several of its variants, including Multi-Paxos, Vertical Paxos, Fast Paxos, Flexible Paxos and Stoppable Paxos. To the best of our knowledge, this work is the first to verify these protocols using a decidable logic, and the first formal verification of Vertical Paxos, Fast Paxos and Stoppable Paxos.



# On the Virtues of Multi-Modal Verification

# Case Study: The Rabia Protocol

- Original formalisation: Ivy + Rocq
- **Ivy** for protocol modelling and automated invariant checking
- **Rocq** for interactive proofs for some additional invariants



**Rabia: Simplifying State-Machine Replication Through Randomization**


Haochen Pan*
University of Chicago
Chicago, IL, USA
haochenpan@uchicago.edu

Jesse Tuglu*
University of Michigan
Ann Arbor, MI, USA
tuglu@umich.edu

Neo Zhou
Boston College
Boston, MA, USA
neo.zhou@bc.edu

Tianshu Wang*
Duke University
Durham, NC, USA
tw295@duke.edu

Yicheng Shen
Boston College
Boston, MA, USA
yicheng.shen@bc.edu

Xiong Zheng†
Google, Inc.
Kirkland, WA, USA
xiongzheng@google.com

Joseph Tassarotti
Boston College
Boston, MA, USA
tassarot@bc.edu

Lewis Tseng
Boston College
Boston, MA, USA
lewis.tseng@bc.edu

Roberto Palmieri
Lehigh University
Bethlehem, PA, USA
palmieri@lehigh.edu



**Abstract**

We introduce Rabia, a simple and high performance framework for implementing state-machine replication (SMR) within a datacenter. The main innovation of Rabia is in using *randomization* to simplify the design. Rabia provides the following two features: (i) It does not need any fail-over protocol

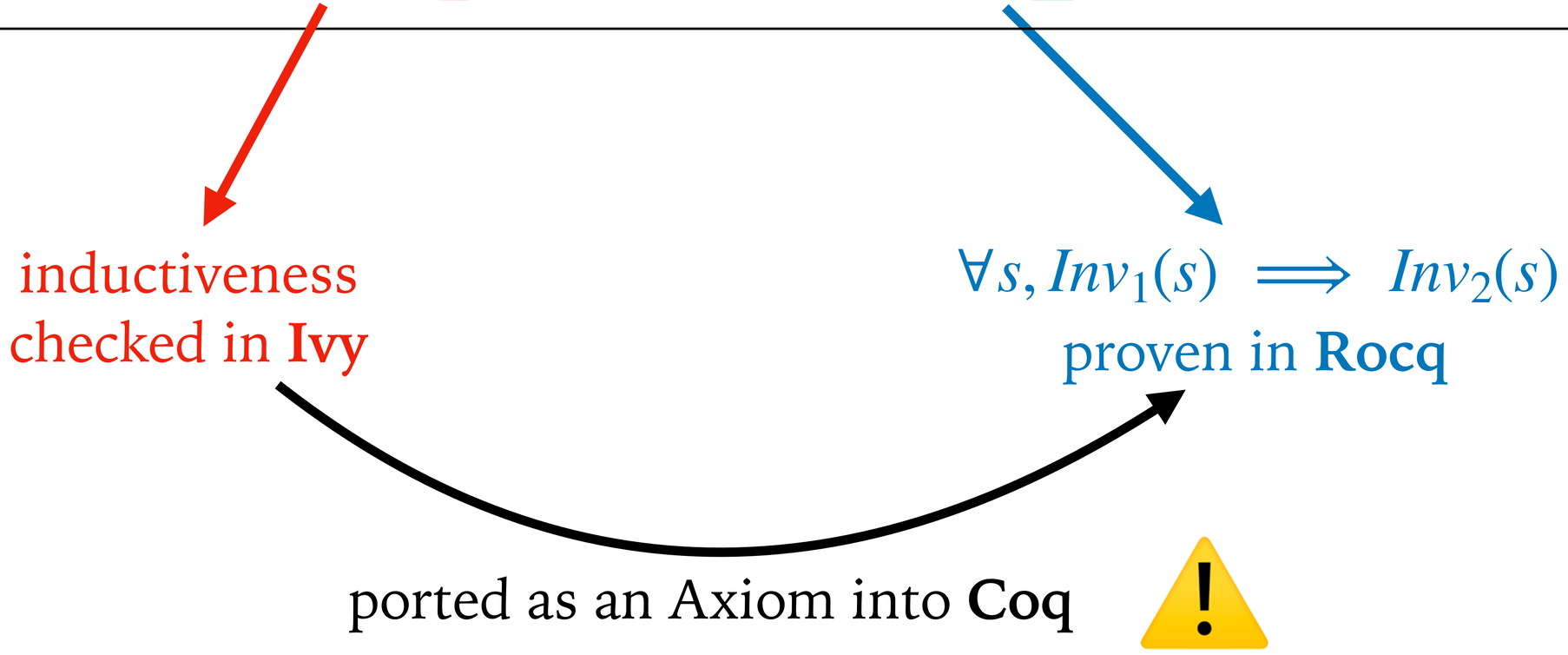


***Keywords:*** SMR, Consensus, Formal Verification

**ACM Reference Format:**
Haochen Pan, Jesse Tuglu, Neo Zhou, Tianshu Wang, Yicheng Shen, Xiong Zheng, Joseph Tassarotti, Lewis Tseng, and Roberto Palmieri. 2021. Rabia: Simplifying State-Machine Replication Through Randomization. In *ACM SIGOPS 28th Symposium on Operating Systems*

# Case Study: The Rabia Protocol

Claim: $Inv_1 \wedge Inv_2$ is an invariant.



# Case Study: The Rabia Protocol

- Moving facts between the provers introduced a discrepancy

*Ivy*

```
specification {
  conjecture [started_pred] (ind_defs.started(Psucc) & succ(P, Psucc) -> ind_defs.started(Psucc))
}
```

*Rocq*

```
Definition started_pred σ :=
  (∀ P, started σ (succ P) → started σ P).
Axiom started_pred_invariant : invariant started_pred.
```

- In **Veil**, when proving $Inv_2$ , the proof of $Inv_1$ (from `#check_invariants`) is reused; this allowed us to catch and fix this discrepancy.

# Future Directions

- Automatically synthesising a FO encoding from concrete implementation
- Better UI for interfacing between automated and interactive proofs
- Support for liveness reasoning (both concrete and symbolic)
- IronFleet-style system verification in Veil + Velvet

# To Take Away

- **Veil** is a Lean library for *automated/interactive* verification of distributed protocols
- Combines *symbolic* proofs and TLC-style *concrete-state* model checker
- *Foundational*: different VC generators are proven sound wrt. concrete semantics
- Acceptable performance for FOL, *seamless integration* with HO specifications

try.veil.dev https://github.com/verse-lab/veil branch “veil-2.0”

*Veil: A Framework for Automated and Interactive Verification of Transition Systems*, CAV’25

*Lessons from Building an Auto-Active Verifier in Lean*, Dafny’26

Thanks!